MaxonC++

Das Entwicklungssystem zur objektorientierten Programmierung

AMIGA

MaxonSDB

Source-Level-Debugger

MAXON
computer

MaxonSDB (Source-Level-Debugger)

Autor: Christian Spreuer

© 1992 by MAXON Computer

Alle Rechte vorbehalten. Dieses Handbuch und die dazugehörige Software ist urheberrechtlich geschützt. Es darf in keiner Form (auch auszugsweise) mittels irgendwelcher Verfahren reproduziert, gesendet, vervielfältigt bzw. verbreitet oder in eine andere Sprache übersetzt werden.

Bei der Erstellung des Programms, der Anleitung sowie Abbildungen wurde mit allergrößter Sorgfalt vorgegangen. Trotzdem können Fehler nicht ausgeschlossen werden. MAXON übernimmt keinerlei Haftung für Schäden, die auf eine Fehlfunktion von Programmen zurückzuführen sind.

Alle Informationen, die in der vorliegenden Anleitungen enthalten sind, werden ohne Rücksicht auf einen eventuellen Patentschutz veröffentlicht. Ebenso werden Warenzeichen ohne Gewährleistung einer freien Verwendung benutzt.

Support
Bei Fragen zu MaxonSDB wenden Sie sich bitte an:

MAXON Computer
Schwalbacher Str. 52
W-6236 Eschborn.

Inhaltsverzeichnis

1. Einführung 9

Über MaxonSDB 9
Was können Sie mit MaxonSDB tun? 10
Was können Sie mit MaxonSDB nicht tun? 10

Über dieses Handbuch 11
Begriffe 11
Typographie 11
Tastatur 12

Mindestinhalt der Disketten 12

2. Die Benutzung von MaxonSDB 13

Die Vorbereitung Ihres Programms 13

Programmstart 13
Kommandozeilenoptionen 14

3. Die Fenster 15
Der Aufbau der Fenster 15
Die Arbeit mit Fenstern 16
Die gewählte Zeile 17
Die verschiedenen Fenstertypen 18
Das Hauptfenster zeigt Quelltext 18
Statusmeldungen 19
Ein Streifen Haltepunkte 19
Quelltext oder Assembler? 20
Die Fenster mit globalen Variablen und Funktionen 21
Die Fenster mit globalen Variablen 22
Die Fenster mit globalen Funktionen 23
Die Fenster mit lokalen Variablen 23
Das Überwachen-Fenster 24
Das Stapel-Fenster 25
Technischer Hintergrund 25
Die Inspiziere-Fenster 27
Inspiziere Skalare, Zeiger und Referenzen 28
Inspiziere Funktionen 28
Inspiziere Felder (Arrays) 28
Inspiziere Klassen 28
Das lokale Menü des Inspect-Fensters 29

Das Modul-Fenster 30
Das CPU-Register-Fenster 30
Das Speicher-Fenster 32
Das Texteingabe-Fenster 34
Die Darstellung von Daten 36

4. Die Menüs 38
Aufbau der Menüs 38
Alle Menübefehle 38
Das Projekt-Menü 38
Programm laden... 38
Info/Über... 39
Hilfe... 39
Debugger beenden 39
Das Zeige-Menü 39
Stapel 40
Aktuelle Var. 40
Globale Var. 40
Funktionen 40
Überwachen 40
Module 41
Speicher 41
CPU-Register 41
Das Bearbeite-Menü 41
Zeige Zeile... 41
Zeige PC 42
Schritt 42
Gehe 42
Folge hinein 42
Schritt über 42
Gehe bis Rückkehr 43
Gehe ohne Haltepkte 43
Stopp 43
Neu Starten 44
Argumente... 44
Das Haltepunkte-Menü 44
Alle löschen 44
Das Konfigurations-Menü 45
Allgemein... 45
Lade Konfiguration... 45
Sichere Konfig. 45
Sichere Konf. als... 45
Das Fenster-Menü 46
Zoom 46
Aktuell 46
Neuzeichnen 46
Schließen 47
Dieses 47
Nächstes 47
Vorheriges 47

Fenster 1 bis Fenster 10 47
Die lokalen Menüs 48
Quelltext 48
Überwachen 48
Inspizieren 49
Absteigen 49
Lokale Var. 49
Speicher 49
Löschen 49
Alles Löschen 50
Neue Adresse... 50
Indexbereich... 50
Adressgrenzen... 50
Daten ändern... 50
als Byte 51
als Word 51
als Long 51
Dezimal 51
Hexadezimal 51
Oktal Basis 2 51
7-Bit-ASCII 51
UniCode 51
Sortieren aus 51
nach Name 51
nach Ort 51
Zeige Quelltext 52
Zeige Assembler 52
Zeige beides 52

5. Haltepunkte 53
Haltepunke setzen 53
Symbole im Hauptfenster 53
Symbolisierung des Programmzählers 54
Symbolisierung der Haltepunkte 54
Bedingte Haltepunkte 55
Das Haltepunkt-Dialogfenster 55

6. Konfiguration 58
Die Datei MaxonSDB.prefs 58
Die Konfigurations-Dialogfenster 59
Allgemeine Parameter 59
Benutzersprache: 59
Quellsprache: 60
Betriebsart: 60
Funktioniere mit: 61
Tabulatorgröße: 61

7. Dialogfenster, Informations- und Fehlermeldungen 62

Meldungen mit Dialogfenster 62
Benötigt residente Libraries! 62
Bitte zuerst ein Modul wählen oder laden! 62
ChipMem-Speichermangel! 63
Datei hat falsches Format! 63
Datei nicht gefunden! 63
Fehlerhafte ".mdbg"Datei! 63
'HotHelp' ist nicht installiert! 63
Ihr Programm wartet derzeit auf Signale. 64
Internal error 64
Ist keine IFF-Datei! 64
Ist keine MDBI-Datei! 65
Kann Datei nicht erzeugen! 65
Keine ausführbare Datei! 65
Kein Programm geladen! 65
MaxonSDB benötigt rct.library version 16! 65
MaxonSDB wirklich beenden? 66
Overlays nicht erlaubt! 66
Programm endete normal. Rückgabewert: ... .66
Programm läuft bereits! 66
Schreibfehler im Speicher! 67
Schreib oder Lesefehler! 67
Speichermangel! 67
Meldungen in der Statuszeile 67
Kein Quelltext verfügbar! 67

8. Anhang 68

The Enforcer 68

inline-Funktionen 69

Technische Einzelheiten 70

Die zwei Prozesse des MaxonSDB 70
Forbid() und Disable() 71
Die Prozessoren 68020,68030 und 68040 71
Die Coprozessoren 68881 und 68882 72
Debuggen in Amiga-Libraries 72
Debuggen von eigenen Libraries 73
Das Abfangen von Prozessor-Exceptions 73
SafeTrace 74

9. Das Format der Debugger-Dateien 76

Der Inhalt der Dateien 76

Das Format der Dateien 77

Das Format der #?.mdbg-Datei 77
Das Format der #?.mdbi-Datei 79
Chunk STRG 80
Chunk TYPE 81
Chunk ENUM 84
Chunk SCOP 85
Chunk HUNK 85
Chunk SRCS 85
Chunk ILST 86
Chunk POSN 87
Die Codierung von Quellpositionen 88
Die Codierung von Codepositionen 89

Index 91

1. Einführung

Über MaxonSDB

MaxonSDB ist ein leicht zu bedienender Quellsprachen-Debugger für alle Commodore-Amiga-Computer.

Er vereinfacht die Entwicklung von funktionsfähigen Programmen wesentlich, da mit ihm Fehler effektiv gefunden werden können. Durch seine intuitive Bedienbarkeit und die Möglichkeit, fast alle Funktionen des Debuggers mit Tastaturkürzeln zu erreichen, ist er sowohl für den Anfänger als auch den Profi eine ideale Hilfe bei der Fehlersuche.

Einige seiner Eigenschaften:

- Läuft auf allen Rechnern der Amiga-Familie mit mindestens 1 MB Speicher.
- Läuft mit allen von Commodore unterstützten Betriebssystemversionen (d.h. mindestens Kickstart 1.3 bis 2.04).
- Läuft auf allen Prozessoren der MC-680x0-Familie, auch mit den Coprozessoren 6888x.
- Hält sich an die von Commodore-Amiga vorgegebenen Richtlinien zur Gestaltung von Benutzeroberflächen.
- Fast vollständig über Tastatur bedienbar.
- Vollständig mit der Maus bedienbar (ausgenommen Texteingabe).
- Frei konfigurierbarer Bildschirmaufbau, beliebig viele Fenster.
- Debuggen auf Quelltext- und Assemblerebene, auch gemischt.
- Komfortable, ausführliche und übersichtliche Darstellung der Datenstrukturen Ihres Programms.
- Voller Zugriff auf Assembler und CPU im User-Modus.
- Unterstützung der objektorientierten Programmierung von C++.
- Automatische Aktualisierung von Fenstern.

Was können Sie mit MaxonSDB tun?

MaxonSDB kann für die Fehlersuche in Programmen verwendet werden, die mit Maxon C/C++ oder KickPascal entwickelt wurden. Reine Assemblerprogramme oder Programme, die von anderen Compilern erzeugt wurden, benötigen ein Konverterprogramm, um die Debuggerinformationsdateien umzukodieren.

MaxonSDB unterstützt Sie bei den beiden schwierigsten und lästigsten Schritten der Programmentwicklung: der Fehlersuche und dem Finden der Fehlerursache.

Mit den mächtigen Befehlen des MaxonSDB können Sie ein Programm im Quelltext an jeder beliebigen Stelle anhalten und Schritt für Schritt ausführen. Dabei haben Sie jederzeit alle oder die für Sie wichtigsten Variablen im Blick, und können diese auch gezielt verändern.

Was können Sie mit MaxonSDB nicht tun?

Auch wenn MaxonSDB äußerst leistungsfähig ist, so kann er doch nicht alles.

Er kann Ihr Programm nicht übersetzen, dazu benötigen Sie einen Compiler wie z.B. MaxonC++ oder KickPascal, der Debuggerinformationen für MaxonSDB erzeugen kann.

MaxonSDB erspart Ihnen nicht das Denken. Er findet Fehler nicht automatisch. Sie müssen sich einen Plan zurechtlegen, wie Sie bei auftretenden Fehlern die Ursachen einkreisen und finden können, denn sonst werden Sie trotz der Unterstützung durch MaxonSDB kaum Zeit sparen.

MaxonSDB fängt nicht alle Abstürze ab. Wenn Ihr Programm wild in den Speicher schreibt und dabei Daten anderer Programme zerstört, so kann MaxonSDB dies nicht verhindern.

Sie können keine Programme debuggen, die das Multitasking abschalten. Dies bedeutet auch, daß Sie keine Interrupt- und Supervisor-Funktionen debuggen können. MaxonSDB kann ein Programm im Supervisor-Modus des Prozessors nicht anhalten, da er ganz normal im Multitasking läuft.

MaxonSDB kontrolliert nur einen Prozess. Sie können keine Programme debuggen, die mehrere Prozesse erzeugen.

Über dieses Handbuch

Begriffe

In diesem Handbuch werden einige Begriffe in einer allgemeineren Bedeutung als üblich verwendet, und für manche Begriffe wird der englische Name verwendet, falls deren deutsche Übersetzung nicht gebräuchlich ist.

Mit **Modul** wird hier eine Übersetzungseinheit bezeichnet, wie sie C oder Assembler-Programmierern bekannt ist. Pascal benutzt hierfür den Begriff „Unit".

Der Begriff **Funktion** deckt sowohl die Funktionen der Sprache C, als auch die Unterprogramme und Prozeduren von Pascal und Assembler ab.

Argumente sind dasselbe wie Parameter. Es wird aber unterschieden zwischen den Kommandozeilenargumenten eines Programms und den Argumenten einer Funktion.

Kickstart: Das Betriebssystem der Amiga-Rechner wird oft unter verschiedenen Namen angesprochen, wobei diese Namen eigentlich nur für einen Teil des Betriebssystems stehen (z.B *AmigaDOS 2.0* oder *Workbench 1.3*). In diesem Handbuch wird einheitlich der Name *Kickstart* für das Betriebssystem des Amiga verwendet. Dabei wird zwischen den Versionen 1.2 und 1.3 nicht unterschieden.

(Der) **Debugger**.
(Der) **Startup-Code**. Hierfür gibt es jeweils keinen gebräuchlichen deutschen Begriff.

Typographie

In diesem Handbuch werden verschiede Schriftarten zur Verdeutlichung von Begriffen und Texten verwendet:

`Schreibmaschine`	Sowohl Textaus- und -eingaben, als auch komplette Programmlistings und im Text stehende Bezeichner aus solchen Listings werden mit dieser Schrift gekennzeichnet.
blablablabla	Hiermit werden Eigennamen hervorgehoben.
„Gänsefüßchen"	Menübefehle werden in Gänsefüßchen eingefaßt.
<Spitze Klammern>	Einzelne Tastendrucke und Tastenkombinationen werden von spitzen Klammern umgeben.

Tastatur

Auch auf die Gefahr hin, Sie hiermit zu langweilen, möchte ich hier kurz die Verwendung von Tastenkombinationen beschreiben, und wie diese im laufenden Text beschrieben sind.

Die Namen von Tasten auf der Tastatur sind im Text mit spitzen Klammern („<" und „>") umgeben. Falls eine Taste einen aufgedruckten Text besitzt, wird dies als Name der Taste verwendet, ansonsten werden folgende Bezeichnungen benutzt: Die vier Cursor-Tasten (halbrechts auf der Tastatur, die Tastengruppe mit den vier Pfeilen) werden mit <Hoch>, <Runter>, <Links> und <Rechts> bezeichnet, und die Tabulatortaste (ganz links Über der <Ctrl>-Taste) mit <Tab>. Die Tasten links und rechts neben der Leertaste sind bekanntlich die <linke Amiga>- und <rechte Amiga>-Taste, und die zwei Tasten mit dicken umrandeten Pfeilen die <Shift>-Tasten.

Eine Tastenkombination besteht aus zwei gleichzeitig gedrückten Tasten. Die Namen der Tasten werden gemeinsam in ein Paar spitze Klammern geschrieben, wobei bei Kombinationen mit der <Ctrl>-Taste diese mit <^> abgekürzt wird. Beispiele für Tastenkombinationen sind: <Amiga X>, <Alt X>, <^X> (statt <Ctrl X>), und <Shift F3>.

Mindestinhalt der Disketten

Auf der Diskette müssen sich folgende Dateien befinden:

- **MaxonSDB**: Der Debugger.
- **MaxonSDB.RCT**: Die Ressourcendatei des Debuggers.
- **MaxonSDB.Prefs**: Eine Konfigurationsdatei des Debuggers.
- **SafeTrace**: Ein Hilfsprogramm, das eventuelle Probleme mit dem Trace-Modus behebt.
- **rct.library** Die Ressourc-Library. V16.24 oder höher.

Genaue Angaben finden Sie, wenn Sie die Readme-Datei auf der Diskette lesen. Da sich nach Drucklegung dieses Handbuches noch Änderungen ergeben können, wählten wir diesen Weg.

2. Die Benutzung von MaxonSDB

Die Vorbereitung Ihres Programms

Um ein Programm mit MaxonSDB zu debuggen, muß dieses dafür vorbereitet sein. Das bedeutet, daß für Ihr Programm Debug-Informations-Dateien erzeugt werden müssen, damit MaxonSDB Quelltext und Programmcode einander zuordnen kann. In der integrierten Programmierumgebung von „MaxonC++“ müssen Sie dazu mit dem Menü-Befehl „Optionen/Compiler“ ein Dialogfenster öffnen, und in diesem Fenster unter der Überschrift „Optionen“ die Option „Debug-Datei“ anwählen, und Ihr Programm dann neu übersetzen. Bei der Kommandozeilenversion des MaxonC++ - Compilers „cppc“ muß bei jedem Aufruf die Option „-b“ angegeben sein.

Andere Compiler erzeugen eventuell ebenfalls Debug-Dateien, die für MaxonSDB geeignet sind. Bitte entnehmen Sie die dafür nötigen Befehle und Schritte dem Handbuch des jeweiligen Compilers.

Programmstart

Sie starten MaxonSDB über das CLI oder die Shell. Ein Start von der Workbench ist derzeit nicht sinnvoll möglich.

Wenn Sie nur `MaxonSDB` eintippen, so können oder müssen Sie den Namen Ihres zu debuggenden Programms mit dem Menübefehl „Projekt/Programm laden...“ angeben, und vorher die Programmargumente mit „Schritt/Argumente...“ setzen.

Sie können beides aber auch direkt in der Komandozeile des MaxonSDB angeben. Wenn Sie das Programm Demo in der Shell oder dem CLI normalerweise mit

```
Demo dies ist ein Test
```

aufgerufen hätten, so setzen Sie einfach noch ein `MaxonSDB` davor, und das Programm wird unter der Kontrolle von MaxonSDB ausgeführt. Wenn Sie an MaxonSDB dabei noch irgendwelche Kommandozeilenoptionen übergeben wollen, so müssen diese vor dem zu debuggenden Programm stehen, da der gesamte Text nach dem Programmnamen von MaxonSDB ignoriert wird und unverändert an Ihr Programm weitergereicht wird. Im obigen Beispiel könnte ein komplexerer Aufruf von MaxonSDB so aussehen:

```
MaxonSDB -a -t2 Demo dies ist ein Test
```

Im folgenden Abschnitt werden alle Kommandozeilenoptionen des MaxonSDB erklärt.

Kommandozeilenoptionen

Alle Konfigurationseinstellungen des MaxonSDB, die Sie mit „Konfiguration/Allgemein..." nach dem Start wählen können, können Sie auch als Kommandozeilenoption angeben.

Der Aufbau der Optionen ist immer gleich. Jede Option beginnt mit einem Minus-Zeichen -, dann folgt der Kennbuchstabe der Option und noch eine Zahl, die aber auch weggelassen werden kann. Beim Kennbuchstaben wird Groß-/Kleinschreibung ignoriert, `-a` und `-A` ist also dasselbe. Wenn die Zahl fehlt, dann ist dies gleichbedeutend wie wenn die Zahl 1 angegeben wäre.

Die Reihenfolge der Optionen in der Zeile ist unwichtig. Die mit der Kommandozeile eingestellten Optionen haben auf jeden Fall Vorrang vor den in den Konfigurationsdateien abgelegten Einstellungen.

Normalerweise dient die Zahl 0 dazu, eine Option wieder auf ihren Standardwert zurückzusetzen.

Es gibt folgende Optionen:

- `-a`: „Assemblermodus" — Starte den Debugger statt im Quellmodus im Assemblermodus, so daß der Startup-Code des Programms nicht automatisch ausgeführt wird und das Hauptfenster ein Disassemblerlisting zeigt.

- `-e`: „Enforcer" — Arbeite besser mit Enforcer V37 zusammen. Wenn diese Option nicht gesetzt wird, dann kann virtueller Speicher benutzt werden. Wenn Sie einen älteren Enforcer benutzen, brauchen Sie diese Option nicht anzugeben. Mehr dazu finden Sie im Anhang im Kapitel „Enforcer".

- `-l`: „Language" — Gib alle Texte auf Englisch aus. Normalerweise gibt MaxonSDB alle Texte auf Deutsch aus, aber Englisch kann er auch, wenn Sie dies bevorzugen sollten.

- `-s`: „Source" — Wenn Sie wollen, daß MaxonSDB die Namen der einfachen Datentypen nicht wie auf dem Amiga üblich mit `ULONG`, `WORD` usw. darstellt, sondern hier dem (ANSI-) Standard folgt und `unsigned long`, `short` usw. schreibt, dann geben Sie diese Option an.

- `-t`: „Tabulator" — Wenn Sie in Ihren Quelltexten Tabulatoren verwenden, und der für diese Tabulatoren eingestellte Wert ist nicht 8, dann können Sie hiermit Ihren bevorzugten Wert setzen. Der hierbei angegebene Wert des Tabulators muß zwischen 1 und 40 einschließlich liegen.

Nach den Optionen folgt eventuell der Name des zu debuggenden Programms und danach dessen Argumente.

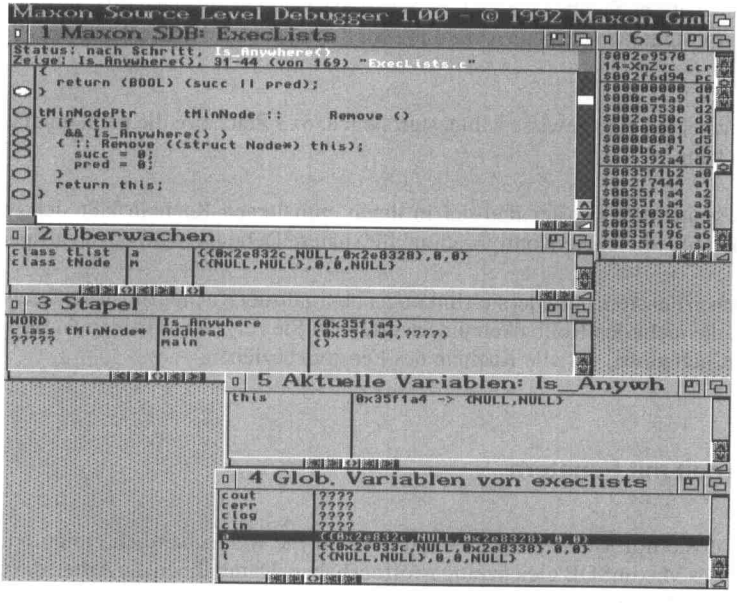
3. Die Fenster

MaxonSDB macht ausgiebigen Gebrauch von der Benutzeroberfläche „Intuition". Jede Information, die MaxonSDB darstellt, erscheint in Fenstern, und jedes Fenster hat ein Menü, Symbole, Rollbalken, usw. Falls Sie bereits mit dem Amiga gearbeitet haben, werden Sie MaxonSDB auf Anhieb benutzen können. Falls Sie auf anderen Rechnern bereits mit modernen Debuggern gearbeitet haben, fällt der Umstieg zu MaxonSDB ebenfalls leicht.

Der Aufbau der Fenster

Fast alle Fenster von MaxonSDB haben den gleichen Grundaufbau, der hier beschrieben ist.

Es gibt Fenster, in denen die Werte von Variablen auf verschiedene Art gezeigt werden, ein Fenster, in dem Quelltext und Haltepunkte gezeigt werden, Fenster mit Blick auf den Speicherinhalt und auf die CPU-Register und noch ein paar andere mehr.



Jedes Fenster des MaxonSDB kann man selbstverständlich mit den üblichen Intuition-Methoden (aber zum Teil auch mit Tastaturkürzeln!) verschieben, vergrößern und verkleinern, nach vorne und hinten stellen und schließen.

Als wichtigste Bestandteile enthält jedes Fenster eine oder mehrere rechteckige Flächen, in denen Daten (meistens Text) dargestellt werden. Diese Flächen nennen wir „Kacheln", weil sie wie Kacheln an der Wand regelmäßig nebeneinander und übereinander angeordnet und durch eine Linie voneinander getrennt sind. Im Englischen gibt es hierfür die Begriffe „Pane" oder „Panel", die aber auf dem Amiga nicht geläufig sind.

Die Daten in den benachbarten Kacheln eines Fensters haben immer etwas gemeinsam, sonst würden sie in verschiedenen Fenstern dargestellt. Deshalb haben alle nebeneinanderliegenden Kacheln eines Fensters die gleiche Höhe und alle übereinanderliegenden Kacheln eines Fensters die gleiche Breite, und der in ihnen sichtbare Datenausschnitt wird jeweils mit einem gemeinsamen Rollbalken rechts oder unterhalb der Kacheln eingestellt.

Ein Rollbalken besteht, wie spätestens seit Kickstart 2.0 üblich, aus einem Schieberegler und zwei Pfeil-Symbolen. Während Sie den Fensterinhalt mit einem Rollbalken verschieben, wird der sichtbare Ausschnitt ständig und sofort aktualisiert. Falls ein Fenster mehr als eine Kachel hat, kann man die Größe aneinandergrenzender Kacheln durch ein spezielles Symbol zwischen den Rollbalken, das „Kachelgrößengadget", einstellen, indem man dieses Symbol anklickt, festhält und verschiebt. Dabei wird ständig mit einer Linie dargestellt, wie die neue Aufteilung der Kacheln nach dem Loslassen des Symbols sein wird.

Eine Kachel hat eine gewisse Mindestgröße (ca. 5x3 Zeichen), die nicht unterschritten werden kann.

Das Aussehen der Fenster und der in ihnen enthaltenen Kacheln läßt sich mit dem Menübefehl „Speichere Konfiguration" für spätere Debugger-Sitzungen abspeichern.

In manchen Fenstern befindet sich links oder oberhalb der Kacheln noch eine rechteckige Fläche, die nicht den Rollbalken unterworfen ist. Sie zeigt im allgemeinen Informationen, die sich global auf alle Kacheln des Fensters beziehen.

Die Arbeit mit Fenstern

Während Sie mit MaxonSDB arbeiten, werden Sie wahrscheinlich einige oder viele Fenster des MaxonSDB gleichzeitig geöffnet haben. Damit Ihnen die Arbeit dabei leicht von der Hand geht, haben wir dafür gesorgt, daß Sie auch ohne die Maus zu verwenden, sehr einfach von einem Fenster zum anderen wechseln und Fenster öffnen und schließen können.

Deshalb hat jedes geöffnete Fenster eine Nummer, die links oben in der Titelzeile direkt neben dem Schließsymbol zu sehen ist. Wenn Sie eine der (oder beide) Alt-Tasten zusammen mit einer der Ziffertasten unterhalb der Funktionstasten drücken, wird das entsprechende Fenster aktiviert und nach vorne gebracht, und alle folgenden Eingaben beziehen sich auf dieses Fenster.

Damit Sie nicht ständig verfolgen müssen, welches Fenster gerade welche Nummer hat, werden die Nummern bereits geöffneter Fenster nicht mehr verändert. Ansonsten erfolgt die Nummerierung der Fenster nach einem sehr einfachen, logischen Schema: Das Fenster, das zuerst geöffnet wird, bekommt die Nummer 1, das nächste die Nummer 2, usw. Wenn eines oder mehrere Fenster wieder geschlossen werden, sind deren Nummern wieder frei, und das Fenster, das dann als nächstes geöffnet wird, erhält die niedrigste freie Nummer. Damit wird sichergestellt, daß die offenen Fenster eine möglichst niedrige Nummer haben und somit per Tastendruck schnell erreichbar sind.

Falls Sie viele Fenster gleichzeitig offen haben, genügen die zehn Tastaturkürzel <Alt 1> bis <Alt 9> und <Alt 0> nicht mehr, um alle Fenster sofort zu erreichen. Aber auch Fenster mit einer Nummer, die größer ist als zehn, sind per Tastatur erreichbar: Mit den Tastaturkürzeln <Alt +> und <Alt -> (oder <F6> und <Shift F6>) wechseln Sie direkt zum Fenster mit der nächsthöheren oder -niedrigeren Nummer, und können so bei mehrfacher Anwendung dieser Kürzel zu jedem offenen Fenster des MaxonSDB gelangen.

Nicht von allen Fenstertypen lassen sich mehrere Exemplare öffnen. Falls Sie ein solches Fenster noch ein zweites Mal öffnen wollen, wird statt dessen das bereits offene Fenster nach vorne gebracht und zum aktuellen Fenster. Dies können Sie auch dazu verwenden, um z.B. direkt das Überwachen-Fenster nach vorne holen, indem Sie den Befehl zum Öffnen dieses Fensters geben („Zeige/Überwachen" <Alt W>), unabhängig davon, ob dieses Fenster schon geöffnet ist oder welche Nummer es gerade hat.

Die gewählte Zeile

Auch innerhalb eines Fensters sind nicht immer alle Kacheln gleichzeitig aktiv und nehmen Ihre Eingaben entgegen. Eine aktive Kachel erkennen Sie daran, daß in ihr eine Zeile hervorgehoben ist, die sogenannte angewählte Zeile.

Wenn in einem Fenster gar keine Zeile hervorgehoben ist, verhält sich MaxonSDB so, als ob (mindestens) die linke obere Kachel des Fensters die aktive Kachel wäre, diese aber keine gewählte Zeile hat.

Die angewählte Zeile zeigt diejenigen Daten, auf die sich die Befehle des lokalen Menüs des Fensters beziehen. Es gibt aber auch Befehle in lokalen Menüs, die sich global auf alle Daten des Fensters beziehen.

Um die gewählte Zeile innerhalb einer Kachel zu wechseln, benutzen Sie einfach die Pfeiltasten <Hoch> und <Runter>. Falls Sie dabei an den Rand der Kachel geraten, rollt MaxonSDB den Kachelinhalt automatisch weiter.

Um eine Zeile in einer anderen Kachel anzuwählen, wechseln Sie zuerst mit <Tab> oder <Shift Tab> in die gewünschte Kachel und dann mit den Pfeiltasten zur gewünschten Zeile.

Sie können die gewünschte Zeile auch einfach mit der Maus anklicken, egal welches Fenster oder welche Kachel gerade aktiv ist.

Jedes Fenster, das dies unterstützt, kann eine gewählte Zeile besitzen, d.h. es können gleichzeitig durchaus mehrere Zeilen angewählt sein, solange jedes Fenster höchstens eine angewählte Zeile hat. Es ist zu jeder Zeit offensichlich und eindeutig, auf welche Zeile sich ein Befehl bezieht, da er sich nur auf die eine angewählte Zeile des einen aktiven Fensters beziehen kann.

Da die Hervorhebung einer angewählten Zeile die Lesbarkeit dieses Fensters etwas herabsetzt, kann man eine gewählte Zeile auch wieder abwählen: Drücken Sie die <Esc>-Taste, und im aktuellen Fenster gibt es keine gewählte Zeile mehr.

Die verschiedenen Fenstertypen

MaxonSDB hat viele verschiedene Fenstertypen, jeder Fenstertyp hat seinen speziellen Zweck. Wenn Sie gelernt haben, welches Fenster wofür geeignet ist, beherrschen Sie den Umgang mit MaxonSDB.

Das Hauptfenster zeigt Quelltext

In diesem Fenster, das den Titel „MaxonSDB“ hat, befindet sich oben eine Kachel mit zwei Zeilen Statusmeldungen und darunter eine (normalerweise recht große) Kachel, in der MaxonSDB Ihnen den aktuellen Quelltext und/oder disassemblierte Maschinenbefehle zeigt.

Es gibt nur ein Hauptfenster. Dieses wird automatisch geöffnet, wenn Quelltext oder Assembler zu zeigen ist. Sie können das Hauptfenster deshalb z.B. mit dem Befehl („Bearbeite/Zeige PC“) öffnen.

Statusmeldungen

In den zwei Statuszeilen dieses Fensters wird der Zustand des Programms und die gezeigte Datei beschrieben.

Die erste Zeile gibt an, was Ihr Programm gerade macht, d.h. ob Ihr Programm auf eine Anweisung von MaxonSDB wartet oder ob es läuft. Wenn Ihr Programm läuft, werden folgende Zustände unterschieden: Ob der Prozeß tatsächlich frei läuft („Läuft: running") oder ihm nur der Prozessor dafür fehlt („Läuft: ready") oder ob der Prozeß derzeit auf ein Ereignis wartet („Läuft: waiting").

Falls der Prozeß auf eine Anweisung von MaxonSDB wartet, werden folgende Meldungen hier erscheinen: Wenn noch kein Programm geladen wurde, wird „nicht vorhanden" ausgegeben. Bevor der erste Maschinenbefehl Ihres Programms ausgeführt wird, steht hier „Neustart", und nach der Ausführung der letzten `RTS`-Anweisung des Programms „beendet". In der Zeit dazwischen wurde entweder ein „Haltepunkt" erreicht oder ein Einzelschritt durchgeführt („nach Schritt"), oder Sie haben die Programmausführung „gestoppt". Oder eine der über fünfzig Ausnahmebedingungen des Prozessors ist aufgetreten (wobei dies zusätzlich mit einem Dialogfenster unübersehbar gemeldet wird). Dann erscheint hier der (englische) Name der Ausnahmebedingung (z.B. „Ausnahme: Zero Divide").

Rechs neben dem Prozeßzustand wird evtl. noch der Name der Funktion angegeben, innerhalb der sich der Programmzähler gerade befindet.

Die zweite Zeile beschreibt, was gerade im Fenster zu sehen ist. Immer wenn im Fenster eine Funktion zu sehen ist, wird deren Name angegeben. (Dies ist oft, aber nicht immer, derselbe Name wie in der Zeile oberhalb). Falls im Fenster nur ein Disassemblerlisting gezeigt wird, wird noch der Text „Assembler ab ..." und die Adresse der ersten Zeile ausgegeben. Wenn aber Quelltext im Fenster sichtbar ist, werden die Nummern der ersten und letzten dargestellten Zeilen, die Gesamtanzahl der Zeilen der Datei und der Dateiname ausgegeben.

Ein Streifen Haltepunkte

Auf der linken Seite des Festers befindet sich ein senkrechter schmaler Streifen. In diesem Streifen werden zu jeder Zeile der Kachel noch ein oder zwei Symbole gezeigt. Die Symbole sagen Ihnen, ob Sie in dieser Zeile einen Haltepunkt setzen können oder schon einer gesetzt ist, und ob der Programmzähler sich in dieser Zeile befindet.

Haltepunkte werden durch verschiedenfarbige Achtecke und der Programmzähler wird durch einen Pfeil symbolisiert.

Ausführliche Informationen dazu finden Sie im Kapitel über Haltepunkte.

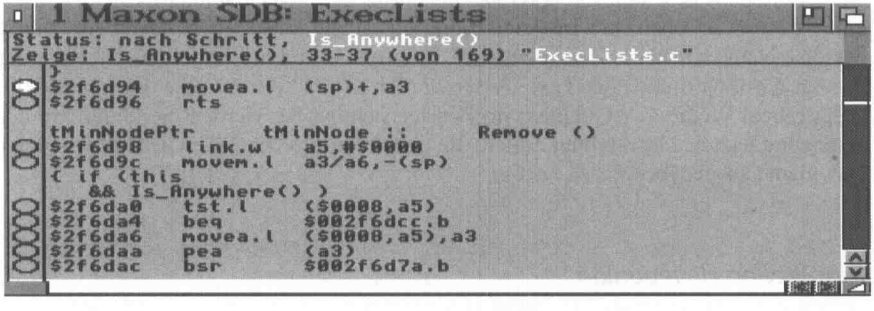
Quelltext oder Assembler?

Normalerweise werden Sie wahrscheinlich nur den Quelltext sehen wollen, aber bei manchen speziellen Problemen kommt man um Debuggen auf Assembler-Ebene nicht herum.

Prinzipiell können Sie wählen, ob MaxonSDB Quelltext oder Assembler oder beides gemischt zeigen soll. Dies geschieht mit den unten besprochenen Befehlen des lokalen Menüs dieses Fensters.

Aber: Falls MaxonSDB zum aktuellen Wert des Programmzählers keinen Quelltext zeigen kann (weil es vielleicht gar keinen gibt?), zeigt er Ihnen immer statt dessen die disassemblierten Maschinenbefehle.
Umgekehrt: Wenn Sie zu einer Variablen die Deklaration sehen wollen („Lokal/Quelltext" im entsprechenden Fenster), zeigt MaxonSDB Ihnen immer den Quelltext und ignoriert Ihren (eventuellen) Wunsch, diese Stelle als Disassemblerlisting sehen zu können.

Wenn MaxonSDB ein Disassemblerlisting zeigt, obwohl Sie „Zeige Quelltext" eingestellt haben, und Sie kennen sich mit Maschinensprache nicht aus, dann führen Sie einfach das Programm weiter schrittweise oder mit „Gehe bis Rückkehr" fort (wenn dies sinnvoll möglich ist). MaxonSDB wird, sobald das Unterprogramm beendet wurde und wieder Quelltext zum PC verfügbar ist, diesen auch zeigen.



Wenn Sie wollen, kann MaxonSDB Ihnen auch Quelltext und Assembler gleichzeitig zeigen: Abwechselnd ein paar Zeilen Quelltext und dann die dazu vom Compiler erzeugten Maschinenbefehle, dann wieder Quelltext usw... Dies ist z.B. nützlich, wenn Sie wissen wollen, was der Compiler aus Ihrem Quelltext gemacht hat.

Im Hauptfenster ist keine Zeile anwählbar, aber Sie können den sichtbaren Fensterinhalt trotzdem ganz leicht mit den Pfeiltasten <Hoch> und <Runter>, <Links> und <Rechts>

verschieben. Falls mindestens eine Quelltextzeile gezeigt wird, rollt der Fensterinhalt anhand der dargestellten Quelltextzeilen und die eventuell auch gezeigten Disassemblerzeilen sind sozusagen nur schmückendes Beiwerk. Wenn aber nur Assemblerzeilen gezeigt werden, so rollt der Fensterinhalt innerhalb Ihres Programms befehlsweise und außerhalb wortweise.

Folgende Befehle befinden sich im lokalen Menü dieses Fensters:

Mit „Zeige Quelltext" „Zeige Assembler" und „Zeige beides" können Sie Einfluß darauf nehmen, ob MaxonSDB in diesem Fenster Quelltext oder ein Disassemblerlisting zeigt, oder eben beides. Mehr dazu finden Sie oben in der Fensterbeschreibung.

„Speicher": Öffne ein Speicherfenster, in dem der aktuell dargestellte Programmabschnitt als Hexdump gezeigt wird.

„Neue Adresse...": Hiermit können Sie eine andere (PC-) Adresse wählen, deren Quelltext gezeigt oder ab der disassembliert werden soll. Diese Adresse wird solange gezeigt, bis Sie das Programm weiter ausführen lassen oder einen der Befehle „Fenster/Aktuell" oder „Bearbeite/Zeige PC" geben, denn dann wird wieder der aktuelle PC Ihres Programms gezeigt.

Die Fenster mit globalen Variablen und Funktionen

Diese beiden Fenstertypen zeigen die globalen Variablen oder Funktionen eines bestimmten Moduls.

Zu jedem Modul kann nur ein Funktionen-Fenster und ein Globale-Variablen-Fenster gleichzeitig geöffnet sein. Sie öffnen ein Funktionen-Fenster mit dem Befehl („Zeige/Funktionen"), und ein Globale-Variablen-Fenster mit dem Befehl („Zeige/globale Vars."). Dabei wird jeweils das zum aktuellen Modul gehörende Fenster gezeigt, so daß Sie evtl. vorher im Modulfenster das gewünschte Modul anwählen müssen.

Die Reihenfolge, in der die Einträge in diesen Fenstern dargestellt werden, können Sie für jedes Fenster neu wählen. Sie haben dabei die Wahl zwischen unsortiert, alphabetisch sortiert und nach Speicheradressen sortiert. Unsortiert bedeutet hier, daß die Zeilen in der Reihenfolge aufgelistet werden, wie sie in der `#?.mdbi`-Datei stehen. Dies entspricht normalerweise der Reihenfolge, wie der Compiler die Deklarationen abgearbeitet hat, und ist somit, falls der Compiler dabei nicht optimiert, der Sortierung nach Speicheradressen zumindest ähnlich.

Die lokalen Menüs dieser Fenster enthalten jeweils fast das Gleiche, außer daß es bei Funktionen natürlich nicht möglich ist, diese zu verändern und somit auch nicht sinnvoll, sie zu überwachen.

Folgende Befehle befinden sich im lokalen Menü dieser Fenstertypen:

„Quelltext“:	Zeige die Deklaration oder Definition des angewählten Objekts. Der Quelltext erscheint im Hauptfenster.
„Inspizieren“:	Öffne ein Inspect-Fenster, das das angewählte Objekt ausführlicher darstellt.
„Absteigen“:	Schließe dieses Fenster und öffne ein Inspect-Fenster mit dem ausgewählten Objekt.
„Sortieren aus?“:	Zeige den Fensterinhalt in der vom Compiler vorgegebenen Reihenfolge.
„Sortieren nach Name?“:	Zeige die Zeilen alphabetisch aufsteigend sortiert. Groß/Kleinschreibung wird ignoriert.
„Sortieren nach Ort?“:	Sortiere die Objekte nach steigenden Hauptspeicheradressen, so daß initialisierte und nicht initialisierte Daten (`DATA` und `BSS`) getrennt angezeigt werden.

Wenn es sich nicht um ein Fenster handelt, in dem nur Funktionen aufgelistet sind, befinden sich noch die folgenden zwei Befehle zusätzlich im lokalen Menü des Fensters:

„Überwachen“: Füge eine neue Zeile mit dem gewählten Objekt in das Überwachen-Fenster ein.

Falls das gewählte Objekt ein Skalar ist, können Sie mit „Daten ändern...“ den Wert dieser Variablen verändern. MaxonSDB öffnet hierzu das Texteingabe-Fenster.

Die Fenster mit globalen Variablen

Alle statischen, d.h. globalen Variablen eines Moduls können hier mit ihren Werten betrachtet werden. Dazu enthält ein solches Fenster zwei Kacheln. In der linken Kachel wird der Name der Variablen und in der rechten Kachel ihr Wert angezeigt. Falls eine globale Variable in diesem Modul nur als `extern` deklariert, aber nicht definiert wurde, wird nur ihr Name, aber nicht ihr Wert angegeben.

Die Fenster mit globalen Funktionen

Alle globalen Funktionen eines Moduls, also alle Funktionen des Moduls, die zu keiner Klasse gehören und die nicht lokal zu einer anderen Funktion definiert sind (dies geht in C und C++ sowieso nicht), werden hier in einer Kachel aufgelistet.

Die Funktionen einer Klasse können Sie in einem Inspect-Fenster finden, das ein Element dieser Klasse zeigt.

Die Fenster mit lokalen Variablen

Es gibt zwei Fenstertypen, die die lokalen Variablen einer Funktion zeigen.

Das Aktuelle-Variablen-Fenster zeigt immer die Variablen der aktuellen Funktion, das ist die Funktion, in der sich der PC gerade befindet. Falls das Programm fortgesetzt wird und die aktuelle Funktion sich ändert, werden automatisch andere Variablen in diesem Fenster gezeigt. Es gibt nur ein solches Fenster. Sie öffnen das Aktuelle-Variablen-Fenster mit dem Befehl „Zeige/aktuelle Var.".

Das Lokale-Variablen-Fenster zeigt die lokalen Variablen einer beliebigen Funktion, die noch nicht zu Ende ausgeführt wurde. Die in diesem Fenster gezeigten Variablen bleiben immer dieselben, auch wenn das Programm voranschreitet. Wenn die Funktion irgendwann zuende ausgeführt ist, löscht MaxonSDB den Fensterinhalt, aber er schließt das Fenster nicht. Es gibt beliebig viele solcher Fenster. Sie öffnen ein Lokale-Variablen-Fenster, indem Sie im Stapel-Fenster oder im Aktuelle-Variablen-Fenster den Befehl dazu geben („Lokal/Lokale Var."). Im Stapel-Fenster muß dazu eine Funktion angewählt sein.

Diese beiden Fenstertypen enthalten je zwei Kacheln. In der linken Kachel wird der Name der Variablen und in der rechten Kachel ihr Wert angezeigt.

Die Reihenfolge, in der die Einträge in diesen Fenstern dargestellt werden, können Sie für jedes Fenster neu wählen. Dies funktioniert exakt so wie bei den oben beschriebenen Globale-Variablen-Fenstern, siehe dort.

Folgende Befehle befinden sich im lokalen Menü dieser Fenstertypen:

„Quelltext":	Zeige die Deklaration oder Definition des angewählten Objekts. Der Quelltext erscheint im Hauptfenster.
„Lokale Var.":	Diesen Befehl gibt es nur im Aktuelle-Variablen-Fenster. Er öffnet ein Lokale-Variablen-Fenster mit dem derzeit hierdrin angezeigten Inhalt.

„Speicher“: Zeige das angewählte Objekt in einem Speicherfenster.

„Überwachen“: Füge eine neue Zeile mit dem gewählten Objekt in das Überwachen-Fenster ein.

„Inspizieren“: Öffne ein Inspect-Fenster, das das angewählte Objekt ausführlicher darstellt.

„Absteigen“: Schließe dieses Fenster und öffne ein Inspect-Fenster mit dem ausgewählten Objekt.

Falls das gewählte Objekt ein Skalar ist, können Sie mit „Daten ändern...“ den Wert dieser Variablen verändern. MaxonSDB öffnet hierzu das Texteingabe-Fenster.

„Sortieren aus?“: Zeige den Fensterinhalt in der vom Compiler vorgegebenen Reihenfolge.

„Sortieren nach Name?“: Zeige die Zeilen alphabetisch aufsteigend sortiert. Groß/Kleinschreibung wird ignoriert.

„Sortieren nach Ort?“: Sortiert die Objekte nach steigenden Hauptspeicheradressen, so daß initialisierte und nicht initialisierte Daten (`DATA` und `BSS`) getrennt angezeigt werden.

Das Überwachen-Fenster

Jede existierende Variable können Sie sich zusammen mit anderen Variablen desselben Gültigkeitsbereichs in einer Liste in einem der vielen Fenster des MaxonSDB zeigen lassen. Dabei kann es schnell soweit sein, daß Sie fünf oder mehr Fenster im Auge behalten müssen, worunter naturgemäß die Übersichtlichkeit leidet. Um dies zu vereinfachen, gibt es das Überwachen-Fenster, in dem Sie jederzeit die gerade wichtigsten Variablen ablegen können und immer im Blick haben.

Um Platz zu sparen und die Übersichtlichkeit zu erhöhen, erhält jede Variable in diesem Fenster nur eine Zeile. Deshalb besteht das Überwachen-Fenster aus drei nebeneinanderliegenden Kacheln: Die linke Kachel zeigt den Typ der Variablen, die mittlere den Namen und die rechte Kachel den Wert.

Wenn Sie eine neue Zeile in dieses Fenster einfügen, können Sie in gewissen Grenzen festlegen, wo diese neue Zeile eingefügt wird. Wenn Sie eine Zeile im Überwachen-Fenster angewählt haben, so wird eine neue Zeile vor dieser Zeile eingefügt, andernfalls wird die neue Zeile nach der letzten vorhandenen Zeile angehängt.

Die Liste der Variablen, die im Überwachen-Fenster gezeigt werden, bleibt auch nach dem Schließen dieses Fensters erhalten und wird nach einem erneuten Öffnen wieder dargestellt. Sie können auch bei geschlossenem Überwachen-Fenster neue Zeilen an diese Liste anfügen.

Das lokale Menü dieses Fensters enthält folgende Befehle:

„Löschen“:	Hiermit wird die gewählte Zeile aus dem Fenster entfernt, so daß dort wieder mehr Platz ist.
„Alles Löschen“:	Alle Variablen werden aus dem Fensters entfernt.
„Quelltext“:	Zeige die Definition der angewählten Variable im Hauptfenster.
„Speicher“:	Ein neues Fenster zeigt den Speicherinhalt der Variablen.
„Inspizieren“:	Öffne ein Inspect-Fenster zu der angewählten Variable.
„Daten ändern...“:	Falls der Typ der gewählten Variablen ein einfacher Datentyp (Skalar) ist, können Sie den Wert der Variablen hiermit ändern.

Das Stapel-Fenster

In diesem Fenster finden Sie die Liste der Funktionen, die in Ihrem Programm aufgerufen wurden, aber noch nicht zuende ausgeführt sind. Der Inhalt dieses Fensters wird immer automatisch erzeugt. Es gibt nur ein Stapel-Fenster, Sie können es öffnen, indem Sie den Befehl „Zeige/Stapel“ aufrufen.

Dieses Fenster ist ähnlich aufgebaut wie das Überwachen-Fenster, es besteht ebenfalls aus drei nebeneinanderliegenden Kacheln. Die linke Kachel zeigt den Typ des Rückgabewerts der Funktion, die mittlere Kachel den Namen und die rechte Kachel die Werte der Parameter der Funktion. Die gerade aktuelle Funktion befindet sich in der ersten Zeile des Fensters und `main()` in der untersten, letzten Zeile.

Technischer Hintergrund

Um die Aufrufkette der Funktionen verfolgen zu können, ist MaxonSDB darauf angewiesen, daß bei **jedem** Eintritt in eine Funktion ein sogenannter „Rahmen auf dem Stapel“ (Stack-Frame) mit dem `LINK A5,#$xxxx`-Maschinenbefehl angelegt wird und dieser Rahmen bei Rückkehr aus der Funktion durch `UNLK A5` wieder entfernt wird. In solch einem Rahmen befindet sich u.a. ein Zeiger auf den vorherigen Rahmen,

so daß MaxonSDB durch Verfolgen dieser Zeigerkette alle Rahmen und somit alle gerade aufgerufenen Funktionen finden kann.

Wenn eine Funktion von einem Compiler übersetzt wurde, wird diese normalerweise immer einen Rahmen anlegen, da dann der Zugriff auf lokale Variablen und Argumente der Funktion einfacher ist. In handgeschriebenen Maschinenprogrammen wird dieses Maschinenbefehlspaar aber selten vorkommen, da diese Unterprogramme normalerweise recht kurz sind und alle lokalen Variablen in Registern Platz finden. Auch können optimierende Compiler unter gewissen Umständen diesen Befehl wegoptimieren. `inline`-Funktionen benutzen den Rahmen der Funktion, in der sie expandiert wurden und haben somit keinen eigenen Rahmen.

Dies ist deshalb wichtig zu wissen, da eine Funktion, die keinen Rahmen angelegt hat, nur solange im Stapel-Fenster erscheint, wie sie die aktuelle Funktion ist, weil MaxonSDB sie andernfalls nicht in der Kette der Rahmen finden kann. Wenn eine Funktion das Register A5 für andere Zwecke verwendet, kann MaxonSDB die Kette der aufgerufenen Funktionen nicht weiter verfolgen, so daß die Informationen im Stapel-Fenster unvollständig sind.

Solange der `LINK`-Maschinenbefehl einer Funktion noch nicht ausgeführt wurde, zeigt MaxonSDB Ihnen die direkte Vorgängerin der aktuellen Funktion nicht, da er deren Rahmen für den der aktuellen Funktion hält. Auch existieren zu diesem Zeitpunkt in der aktuellen Funktion noch keine lokalen Variablen, so daß MaxonSDB Ihnen hier etwas Falsches präsentieren kann. Wenn Sie zu diesem Zeitpunkt den Wert einer lokalen Variablen ändern wollen, so ändern Sie statt dessen irgendwas im Speicherbereich der lokalen Variablen der Vorgänger-Funktion, und überschreiben dort eventuell wichtige Daten.

Verändern Sie den Wert von lokalen Variablen einer Funktion also nur dann, wenn der Programmzähler-Pfeil nicht auf den Kopf oder auf das Ende der Funktion zeigt. Mit dem Befehl „Lokal/Zeige Beides" im Hauptfenster können Sie sich im Zweifelsfall genau zeigen lassen, ob der `LINK`- Befehl schon bzw. der `UNLK`-Befehl noch nicht ausgeführt wurde.

Im lokalen Menü des Stapel-Fensters gibt es zwei Befehle:

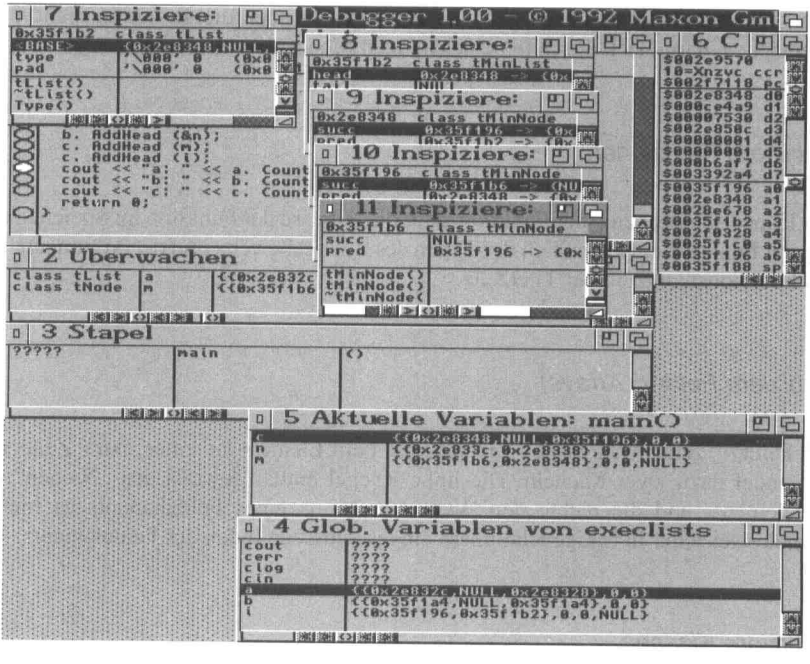
„Quelltext" zeigt Ihnen im Hauptfenster den Quelltext der Funktion der angewählten Zeile.

„Lokale Var." öffnet ein Fenster, in dem die Argumente und lokalen Variablen der angewählten Funktion mit Typ und Name zu sehen sind.

Die Inspiziere-Fenster

Wenn Sie eine Variable oder Funktion ganz genau betrachten wollen oder Ihnen die kompakte einzeilige Darstellung zusammengesetzter Datentypen in anderen Fenstern nicht genügt, können Sie die Variable in einem oder mehreren Inspect-Fenstern ausführlich und erschöpfend untersuchen.

Ein Inspect-Fenster zeigt Ihnen den Namen, die Adresse, den Typ und den Wert des Objekts und bei zusammengesetzten Variablen (das sind Felder und Klassen) die Namen und Werte der darin enthaltenen Objekte. Die Inspect-Fenster für Funktionen, Skalare, Zeigertypen, Felder oder Klassen haben jeweils ein anderes Aussehen und Format, um die unterschiedlichen Daten geeignet darstellen zu können.



Allen Inspect-Fenstern ist gemeinsam, daß sie in der Titelzeile den Text „Inspiziere:" und dann, falls vorhanden, den Namen des Objekts oder zumindest dessen Adresse zeigen. Weiterhin hat jedes Inspect-Fenster oberhalb der Kacheln noch eine Zeile Text, in der die Adresse des Objekts und der Datentyp steht. Die Anzahl und der Inhalt der Kacheln ist abhängig vom Typ des dargestellten Objekts.

Da Inspect-Fenster die Angewohnheit haben, oft in Gruppen aufzutreten, können Sie alle gerade offenen Inspect-Fenster mit dem Tastenkürzel <Shift-F3> automatisch schließen. Hierfür gibt es keinen entsprechenden Menübefehl. Dieser Befehl ist z.B. dann hilfreich, wenn Sie den Aufbau einer komplex verzeigerten Datenstruktur mit Hilfe von Inspect-Fenstern untersucht haben, und nun die Ausführung Ihres Programms fortsetzen wollen.

Inspiziere Skalare, Zeiger und Referenzen

Skalare sind einfache Datentypen wie `char, int, unsigned long, enum x`, usw. Diese Inspect-Fenster enthalten zwei jeweils einzeilige Kacheln. In der linken Kachel wird der Name der Variablen gezeigt und in der rechten ihr Wert. In diesen einzeiligen Kacheln brauchen Sie z.B. den Zeiger oder die Referenz nicht extra anzuwählen, damit sich die Befehle des lokalen Menüs darauf beziehen.

Inspiziere Funktionen

Das Inspect-Fenster zu einer Funktion zeigt die Adresse der Funktion im Speicher, den Typ der Funktion einschließlich der Typen des Rückgabewerts und der Argumente und den Funktionsnamen in der Titelzeile.

Inspiziere Felder (Arrays)

Bei Feldern zeigt MaxonSDB Ihnen jedes Feld-Element in einer extra Zeile und verwendet dazu zwei Kacheln. Die linke Kachel enthält jeweils den „Namen" des Feldelements und die rechte den Wert. Der Name eines Feldelements ist hierbei normalerweise der in eckige Klammern eingeschlossene Index des Feldelements.

Inspiziere Klassen

Ein Inspect-Fenster zu einem Klassenobjekt zeigt die Namen und Werte der Komponenten und Basisklassen in den ersten zwei Kacheln. Falls die Klasse statische Komponenten enthält, werden deren Namen und Werte in zwei zusätzlichen Kacheln gezeigt. Falls die Klasse schließlich Funktionen enthält, werden deren Namen und der Typ ihrer Argumente in weiteren zwei Kacheln gezeigt. Ein Klassenobjekt wird also in einem Inspect-Fenster in zwei, vier oder sechs Kacheln präsentiert, je nach der Art der vorhandenen Komponenten.

Das lokale Menü des Inspect-Fensters

Das lokale Menü der Inspect-Fenster enthält immer die gleichen Befehle, aber manche davon sind bei bestimmten Fenstern dauerhaft abgeschaltet. Z.B. kann eine Funktion nicht verändert werden, so daß hier „Daten ändern..." gesperrt ist.

„Quelltext" zeigt die Deklaration des inspizierten Objekts im Hauptfenster.

„Speicher" öffnet ein Speicherfenster ab der Adresse des inspizierten Objekts.

„Überwachen" fügt eine neue Zeile im „Überwachen"-Fenster hinzu, in der das inspizierte Objekt steht.

„Inspizieren": Wenn Sie eine Komponente des Fensters angewählt haben, so können Sie diese in einem weiteren Inspect-Fenster genauer betrachten. Das neue Fenster öffnet sich genau unterhalb der angewählten Komponente, so daß diese weiterhin sichtbar bleibt.

„Absteigen": Falls Sie eine Komponente eines Objekts genauer betrachten wollen, aber das Objekt selbst Sie nicht weiter interessiert (z.B. bei Zeigern auf Klassen), wählen Sie diesen Befehl. Damit „inspizieren" Sie die gewählte Komponente im gleichen Fenster wie das sie enthaltende Objekt, d.h. Nummer und Ort des Fensters werden beibehalten. Natürlich wird sich der Aufbau des Fensters dem Typ der Komponente anpassen.

„Daten ändern..." Wenn das im Inspect-Fenster dargestellte Objekt ein Skalar ist, können Sie dessen Wert hiermit verändern. Dieser Befehl öffnet das Texteingabe-Fenster, in dem Sie den neuen Wert eingeben können.

Den Wert zusammengesetzter Objekte können Sie nicht direkt als Ganzes ändern. Sie können ein zusammengesetztes Objekt nur so ändern, daß Sie dessen Komponenten einzeln ändern. Dazu müssen Sie, wenn Sie dies nicht bereits getan haben, ein Inspect-Fenster zu diesem Objekt öffnen und darin die einzelnen zu ändernden Komponenten wählen. Eventuell müssen Sie diese Komponente wieder in einem Inspect-Fenster darstellen, usw..., bis Sie schließlich eine einfache Variable vor sich sehen, die Sie dann ändern können.

„Neue Adresse..." Wenn Sie Daten an einer anderen Speicheradresse mit dem Typ dieses Inspect-Fensters betrachten wollen, so können Sie mit diesem Befehl die Basisadresse der im Fenster dargestellten Daten ändern. Dieser Befehl ist notwendig, um z.B. die Daten, auf die ein `void*` zeigt, darstellen zu können, da MaxonSDB Ihnen derzeit keine Möglichkeit bietet, hierfür den gewünschten Datentyp einzugeben.

Das Modul-Fenster

In diesem Fenster zeigt MaxonSDB Ihnen, aus welchen Modulen Ihr Programm besteht. Es gibt nur ein Modul-Fenster, und Sie können es mit dem Befehl „Zeige/Module" öffnen.

Die Module in diesem Fenster sind immer alphabetisch sortiert, und der eigentliche Modulname innerhalb des Pfades wird hervorgehoben. Wenn Sie diese hervorgehobenen Modulnamen nicht auf Anhieb sehen können, weil Sie z.B. tief verschachtelte Verzeichnisse auf Ihrer Festplatte haben, müssen Sie den Inhalt des Fensters mit dem Rollbalken nach links rollen, bis Sie erkennen können, in welcher Zeile welches Modul steht.

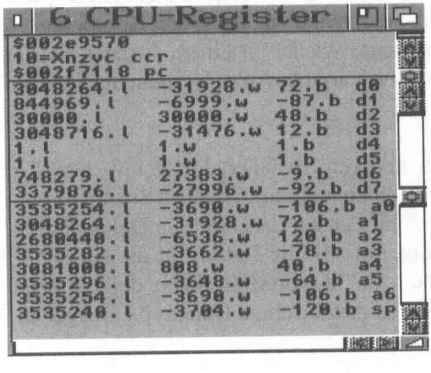
Wenn Sie in diesem Fenster ein Modul anwählen, können Sie sich die globalen Variablen oder Funktionen des Moduls zeigen lassen, indem Sie den Menüpunkt „Zeige/Globale Variablen" oder „Zeige/Funktionen" anwählen. Wenn für MaxonSDB nicht anderweitig klar ist, auf welches Modul sich diese beiden Befehle beziehen, wird er Sie auffordern, zuerst ein Modul zu wählen, und öffnet dazu das Modul-Fenster automatisch.

Es befindet sich nur ein Befehl im lokalen Menü dieses Fensters:

„Quelltext": Hiermit wird die Haupt-Quelldatei des angewählten Moduls im Hauptfenster gezeigt.

Das CPU-Register-Fenster

Manchmal reicht es nicht aus, einen Fehler nur im Quelltext zu suchen. Dann können Sie MaxonSDB z.B. in den Assembler-Modus umschalten, so daß im Hauptfenster ein Disassemblerlisting erscheint. Zusätzlich kann MaxonSDB die Register des untersuchten Tasks in einem sogenannten CPU-Fenster zeigen.



Zu den Registern eines Prozesses gehören nur die Register, die im User-Modus zugänglich sind. Die anderen Register gehören zum Betriebssystem (also zu Exec) und sollten von Ihrem Programm normalerweise nicht gelesen oder gar verändert werden.

Es gibt nur ein CPU-Fenster, und Sie öffnen es, indem Sie den Befehl „Zeige/CPU-Register" geben.

Ein CPU-Fenster besteht aus drei senkrecht angeordneten Kacheln. In der oberen Kachel zeigt MaxonSDB die Adresse des Prozesses (`struct Process`) Ihres Programms, das Statusregister und den Programmzähler. In der mittleren Kachel befinden sich die acht Datenregister und in der unteren Kachel die acht Adressregister einschließlich des Stapelzeigers. Somit haben Sie alle Register des Prozessors, die im User-Modus zugänglich sind, immer im Blick.

Die Prozeßadresse in der ersten Zeile können Sie im allgemeinen ignorieren, da normalerweise der Inhalt der Prozeßstruktur Ihres Programms für die Fehlersuche unwichtig ist.

Ansonsten können Sie diese Adresse benutzen, um ein Inspect-Fenster, das einen `struct Process` zeigt, darauf zu legen, das Ihnen dann alle Informationen über die Prozeßstruktur zeigt. Die Vorraussetzung hierfür ist, daß eines der Module Ihres Programms die Datei `exec/Tasks.h` einbindet und z.B. eine globale Variable vom Typ `struct Process *` definiert. Inspizieren Sie diese Variable, und dann das worauf diese zeigt (z.B. mit „Lokal/Absteigen"), und ändern Sie dann die im Inspect-Fenster gezeigte Adresse auf die hier im CPU-Fenster angegebene Prozeßadresse. Jetzt zeigt das Inspect-Fenster die Prozeßstruktur Ihres Programms. Zugegeben, dies ist etwas umständlich. Aber nur so ist es möglich, daß MaxonSDB Ihnen immer die aktuelle Version der Prozeßstruktur zeigt, auch wenn sich bei zukünftigen Kickstart-Versionen daran etwas ändern sollte.

Die zweite Zeile der obersten Kachel zeigt den Wert des Prozessorstatusregisters CCR. Hier steht der Wert sowohl als zweiziffrige Hex-Zahl als auch aufgeschlüsselt in die fünf definierten Bits X,N,Z,V und C. Falls ein Bit gesetzt ist, so erscheint hier der Großbuchstabe, andernfalls der Kleinbuchstabe. Durch einen Doppelklick auf eines der fünf Bits können Sie dessen Wert schnell invertieren.

Alle anderen Zeilen in den Kacheln dieses Fensters zeigen den Wert und Namen eines Registers, d.h. eine achtziffrige Hex-Zahl und einer der Namen PC, D0 bis D7, A0 bis A6 und SP. Falls Sie eine der Zeilen doppelklicken, so öffnet sich ein Texteingabe-Fenster, in dem Sie einen neuen Wert für das hier dargestellte Register angeben können.

Die gezeigten Werte der Register und Flags sind nur dann aktuell, wenn der Prozeß auf eine Anweisung des MaxonSDB wartet, da MaxonSDB andernfalls keinen Zugriff auf die aktuellen Werte hat und diese sich eventuell auch ständig ändern würden.

Da in diesem Fenster nicht nur ganze Zeilen angewählt werden können, sondern auch einzelne Zeichen, können Sie hier auch mit den Pfeiltasten <Links> und <Rechts> das gewünschte Objekt wählen, wobei aber diese Tasten hier exakt das Gleiche machen wie die Pfeiltasten <Hoch> und <Runter>.

Das lokale Menü dieses Fensters enthält die folgenden Befehle:

„Quelltext" zeigt ihnen im Hauptfenster ein Disassemblerlisting zum aktuellen Wert des Programmzählers. Wenn Sie den Quelltext zum Programmzähler sehen wollen, benutzen Sie besser den Befehl „Bearbeite/Zeige PC", da dieser das Hauptfenster nicht in den Assemblermodus schaltet.

„Speicher" zeigt den Speicherinhalt ab der Adresse im angewählten Register.

„Daten ändern..." invertiert das angewählte Bit des Statusregisters oder öffnet, falls Sie ein Register angewählt haben, das Texteingabe-Fenster, so daß Sie einen neuen Wert für das Register eingeben können.

„Dezimal", „Hexadezimal", „Oktal", „Basis 2", „7-Bit-ASCII", „UniCode": Hiermit wählen Sie, wie der Inhalt aller Register dargestellt wird. Normalerweise werden Sie diese Einstellung fest auf hexadezimal stehen lassen, aber in speziellen Fällen ist diese Wahlmöglichkeit sinnvoll.

Wenn „Dezimal" gewählt ist, wird der Inhalt der Register sowohl als komplettes dezimales Langwort, als auch als dezimales Wort und dezimales Byte ausgegeben (auch bei Adressregistern!), da MaxonSDB nicht wissen kann, wieviel vom Register derzeit gültig ist, und die drei Zahlen oft keinerlei Ähnlichkeit haben. Im Dualsystem und bei Hex-Zahlen ist dieser Aufwand nicht nötig. Eine Hex-Zahl erkennen Sie an dem vorangestellten Dollarzeichen („$"), eine Oktal-Zahl an der vorangestellten Null („0"), und eine Dualzahl fängt mit einem Prozentzeichen („%") an. Dies gilt unabhängig von der benutzten oder eingestellten Quellsprache.

Bei „7-Bit-ASCII" nimmt MaxonSDB an, daß jedes Register vier Standard-ASCII-Zeichen enthält, und diese vier werden ausgegeben. Bei „UniCode" enthält ein Register genau ein 32-Bit langes Zeichen, und nur dieses wird ausgegeben. Zum Thema Textausgabe und Unicode finden Sie beim Speicherfenster genauere Erläuterungen.

Das Speicher-Fenster

Wenn man ein Programm auf Assemblerebene untersuchen will, darf ein Speicherfenster nicht fehlen. In diesen Fenstern wird der Inhalt des Hauptspeichers Byte für Byte im immer gleichen Format dargestellt, nichts wird ausgelassen. Es gibt beliebig viele Speicherfenster, die entweder durch den Befehl „Zeige/Speicher" oder einen der lokalen Befehle „Lokal/Speicher" geöffnet werden.

Ein Speicher-Fenster besteht aus einer Kachel, die nur rechts einen Rollbalken hat, denn ein Rollbalken unten ist überflüssig. Links in dieser Kachel wird in jeder Zeile die

Adresse des ersten darin gezeigten Datenbytes angegeben, und rechts davon folgen nur soviele Daten, wie im Fenster Platz haben (jedoch mindestens ein Datum). In der nächsten Zeile folgen dann weitere Daten ab der direkt folgenden Adresse.

Die Daten können sowohl Dezimal, Hexadezimal, Binär, Oktal, als auch als 7-Bit-ASCII-Text (Standard-ASCII) und 8-Bit-ASCII-Text (d.h. inklusive Latin 1 für Umlaute usw.) dargestellt werden. Dabei können die Daten byte-, wort- oder langwortweise gebündelt gezeigt werden, wobei diese Auswahl bei ASCII-Text (derzeit noch) wenig Sinn macht (und auch ignoriert wird). Sobald Commodore aber den (ganz neu international standardisierten) Unicode wirklich unterstützt (das ist die Erweiterung von ASCII mit 32 oder 16 Bit statt 8 oder 7 Bit), wird dies von MaxonSDB auch unterstützt werden. Zum Glück war Commodore-Amiga damals so schlau, sich beim (8-Bit-) Zeichensatz an die Norm „Latin 1" (ISO-646) zu halten, und nicht was Eigenes zu definieren wie IBM, Atari, Mac oder NeXT, so daß keine Kompatibilitätsprobleme bestehen - außer dem doppelten Platz für ein Zeichen, natürlich.

Außer mit dem Rollbalken rechts kann man den sichtbaren Speicherausschnitt mit den Pfeiltasten <Links>, <Rechts>, <Hoch> und <Runter> verschieben. <Links> und <Rechts> verschieben um jeweils ein Zeichen, Byte, Wort oder Langwort (je nach dem, was gerade gezeigt wird), und <Hoch> und <Runter> um eine ganze Zeile. Wenn man dann genug herumgerollt hat, sich womöglich im Speicher verirrt hat und wieder den Speicher an der ursprünglichen Adresse sehen will, geht das mit <Esc>. Beim Rollen gibt es eine Grenze, die nicht überschritten werden kann: Eine Adresse kleiner als Null gibt es nicht, und eine Adresse größer als 0xFFFFFFFF (4294967295) auch nicht, den Adressen sind bekanntlich vorzeichenlos (ULONG)! Aus technischen Gründen kann MaxonSDB den Inhalt der Speicheradressen 0xFFFFFFF0 bis 0xFFFFFFFF nicht zeigen, aber das ist auch nicht nötig und wird es wohl auch nie sein.

Es kann sein, daß die Adressen 0x80000000 bis 0xFFFFFFFF nie in einem Amiga benutzt werden, da dann viele, viele Programme Probleme bekommen, die mit Adressen vorzeichenbehaftet rechnen. Das ist zwar von Commodore schon lange verboten, aber... Die Probleme sind ähnlich wie damals, als es plötzlich nicht nur CHIP-, sondern auch FAST-Memory gab; und auch als der Amiga 3000 plötzlich Speicher oberhalb der ersten 16 Megabytes hatte... Alle Programmierer wußten, daß dies irgendwann kommen würde, und trotzdem haben sich viele nicht daran gehalten. Diesmal ist es allerdings anders: selbst Commodore hält sich nicht komplett daran, denn es gibt einige (wenige!) dokumentierte Betriebssystemfunktionen, die das höchste Adressbit offiziell für eigene Zwecke verwenden (z.B. AllocEntry). Was kann man daraus schließen? Ja, meinen Sie? Sind 2048 Megabyte Adressraum wirklich weniger wert als ca. 3 selten genutzte Betriebssystemfunktionen?

Das lokale Menü dieses Fensters enthält die folgenden Befehle:

„Neue Adresse..." ermöglicht Ihnen, die erste im Fenster gezeigte Adresse zu wählen, indem Sie sie als Zahl eingeben können.

„als Byte", „als Wort", „als Long":
Hiermit können Sie wählen, in welchen Größen die Daten aus dem Speicher gelesen werden, die dann dargestellt werden (übrigens: ungerade Adressen sind für MaxonSDB nie ein Problem!). Wenn Sie sich Text zeigen lassen und „als Word?" oder „als Long?" gewählt haben, so geht MaxonSDB davon aus, daß Sie in Ihrem Programm tatsächlich einen 16-Bit bzw. 32-Bit breiten Zeichensatz verwenden. Falls dies aber nicht der Fall ist, werden Sie Ihre Texte nicht wiedererkennen...

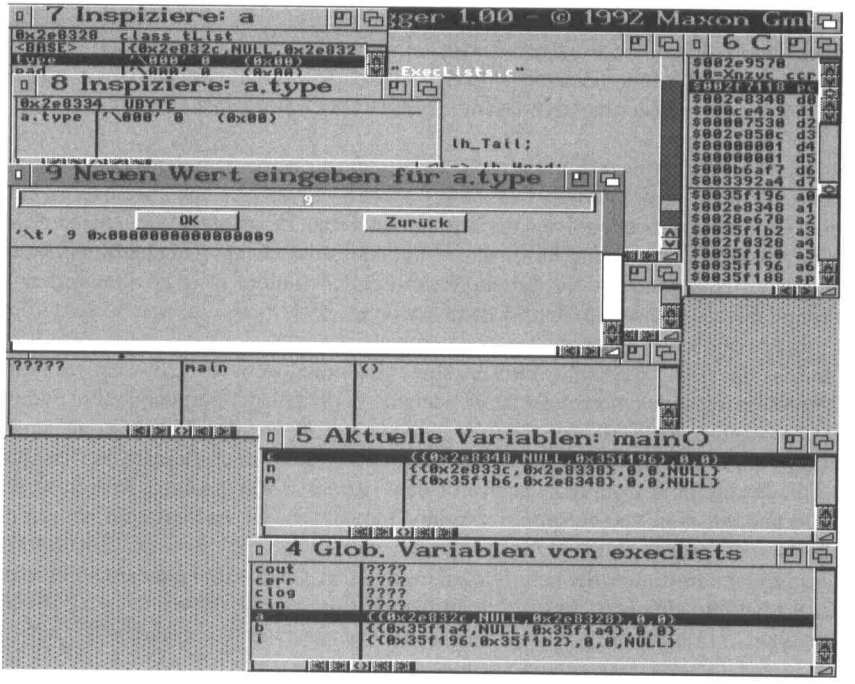
„Dezimal", „Hexadezimal", „Oktal", „Basis 2", „7-Bit-ASCII", „UniCode": Hiermit wählen Sie, wie der Wert eines Datums dargestellt wird. Die ersten vier Optionen sind offensichtlich, aber die beiden anderen bedürfen einer Erklärung: „7-Bit-ASCII" bedeutet, daß nur für die Werte 32 bis 126 das entsprechende Zeichen ausgegeben wird, und für alle anderen Werte ein Punkt „." erscheint, so daß z.B. Umlaute wie ÄÖÜ usw. nicht im Fenster erscheinen. Dies ist nützlich, wenn man im Speicher nach Text suchen will. „UniCode" gibt an, daß alle Bits für die Auswahl des Zeichens von Bedeutung sind. Solange der Unicode von Commodore aber nicht ausreichend unterstützt wird (es gibt noch keine Fonts und Ausgaberoutinen!), kann MaxonSDB nur die Zeichen 32 bis 126 und 160 bis 255 als solche ausgeben, alle anderen erscheinen als Punkt. Diese Zeichen sind der ganz normale Amiga-Zeichensatz. Derzeit sollten Sie, wenn Sie Text darstellen wollen, oben immer „als Byte" wählen (d.h. gewählt lassen, denn MaxonSDB schaltet automatisch auf „Byte", wenn ASCII oder UniCode gewählt wird), da Sie sonst Ihren Text vor lauter Punkten nicht finden werden.

Das Texteingabe-Fenster

Manchmal ist es notwendig, daß Sie eine Zahl oder Text mit der Tastatur eingeben können. Hierfür verwendet MaxonSDB statt eines gewöhnlichen, alles blockierenden Dialogfensters ein ganz normales Fenster mit all den guten Eigenschaften der anderen Fenster von MaxonSDB. Nur ist der Inhalt dieses Fenster für die Texteingabe spezialisiert, so daß Sie es auch sofort erkennen können.

Dieses Fenster bietet eine History-Funktion (Geschichte), damit Sie möglichst keine Zahl oder keinen Text ein zweites Mal eintippen müssen.

Da das Texteingabe-Fenster ein ganz normales Fenster ist, können Sie dieses Fenster verschieben, vergrößern und verkleinern, usw., so daß Sie den Inhalt der anderen Fenster jederzeit erreichen können. Alle Tastenkürzel funktionieren weiterhin. Sie können auch während ein Texteingabe-Fenster offen ist, weitere Fenster öffnen, in denen Sie sich zusätzliche Informationen zeigen lassen.



Sie können sogar das zu debuggende Programm weiterlaufen lassen oder das Fenster schließen, für das Sie im Texteingabe-Fenster eine Zahl eingeben. Allerdings wird MaxonSDB sich dann mit einem Aufblitzen des Bildschirms bei Ihnen beschweren, wenn Sie später das Texteingabe-Fenster mit „OK" verlassen, weil er die eingegebene Zahl nicht mehr verwerten kann.

Es gibt viele Texteingabe-Fenster, z.B. für die Eingabe von Adressen, Zahlen, Fließkommazahlen, Zeilennummern, Programmargumenten... Jedes davon besitzt eine eigene History, da die darin eingetippten Texte normalerweise zu verschieden sind. Zu einer Zeit ist aber höchstens ein Texteingabe-Fenster geöffnet.

Das Texteingabe-Fenster ist folgendermaßen aufgebaut: In der Titelzeile sehen Sie den Zweck des Fensters, den Grund, warum Sie das Fenster geöffnet haben, z.B. „Neue Programmargumente" oder „Neuen Wert eingeben für...".

Unter dem Titel befindet sich ein Texteingabefeld mit dem (spätestens seit Kickstart 2.0) üblichen Aussehen und Verhalten. Darunter befinden sich zwei großflächige Symbole, die die Namen „OK" und „Zurück" tragen. Wenn das Texteingabe-Fenster dazu da ist, daß Sie eine Zahl eingeben können, folgt darunter noch eine Zeile Text, in der MaxonSDB den von Ihnen im Texteingabefeld eingegebenen Text interpretiert und den daraus berechneten Wert auf verschiedene Art darstellt, so daß Sie jederzeit wissen,

welchen Wert der eingegebene Ausdruck hat. Sie können ein solches Fenster also auch als primitiven Taschenrechner verwenden, der ihnen z.B. die Zahl 21439408327765 in Hexadezimal umrechnet (ergibt 0x0000137fc0514c55), Sie müssen dabei nur aufpassen, daß Sie das Fenster nicht versehentlich mit „OK" verlassen...

Unten im Fenster befindet sich noch eine Kachel, die die bisher von Ihnen eingegebenen Texte enthält; bei Programmstart ist diese Kachel natürlich leer. Sie können eine Zeile dieser Kachel wie gewöhnlich mit der Tastatur oder Maus anwählen. Wenn Sie die Zeile mit der Maus anwählen, wird ihr Inhalt sofort in das Texteingabefeld übernommen, bei Tastaturbedienung müssen Sie hierzu <Return> oder <Enter> drücken. Der vorher im Texteingabefeld enthaltene Text ist dann verloren.

Wenn Sie das „OK"-Symbol anklicken, so wird der eingegebene Text, sofern er gültig ist, für die vorgesehenen Zwecke verwendet und in die History aufgenommen, und das Fenster geschlossen. Falls Sie „Zurück" wählen, wird nur das Texteingabe-Fenster geschlossen, und der eingegebene Text wird ignoriert. Sie können diese beiden Symbole (wie in Dialogfenstern üblich) auch mit den Tastenkürzeln <Amiga B> (entspricht „OK") und <Amiga V> (entspricht „Zurück") anwählen, wobei hier natürlich nur die linke Amiga-Taste gemeint ist. Zusätzlich können Sie das „OK"-Symbol mit <^Return> oder <^Enter> anwählen. Sie können das Fenster auch ganz normal mit dem Schließfeld oder <Alt F3> schließen, dies hat denselben Effekt wie die Anwahl des „Zurück"-Symbols.

Wenn Sie in einem Texteingabe-Fenster eine Zahl eingeben sollen, dann läßt MaxonSDB folgende Zahlenformate zu:

Hexadezimalzahlen, beginnend mit „0x" oder „0X" oder „$", Oktalzahlen mit einer „0". Zahlen im Zweiersystem beginnen mit einem Prozentzeichen. Die normalen Dezimalzahlen beginnen mit einer der Ziffern „1" bis „9". Fließkommazahlen haben das übliche Aussehen („-1.0e+5" usw...). Sie dürfen auch für eine Integer-Variable einen Fließkommawert eingeben und umgekehrt. Wenn MaxonSDB den im Texteingabe-Fenster stehenden Text einmal nicht interpretieren kann, so zeigt er in der Zeile darunter fünf Fragezeichen. Die seltsame Art, mit der in der Intel-Welt Zahlen eingegeben werden, wird von MaxonSDB nicht unterstützt (dort wird an eine Ziffernfolge hinten ein „h" oder „b" angehängt, worunter die Lesbarkeit stark leidet).

Die Darstellung von Daten

In vielen Fenstern werden die Werte von Variablen gezeigt. Dabei erfolgt die Darstellung dieser Daten in einem einheitlichen Schema, wobei möglichst viel Information möglichst klar dargestellt wird. Je nach Typ der Daten wird ein unterschiedliches Format benutzt, um deren Wert möglichst informativ in einer Zeile darstellen zu können.

Der Wert eines unbekannten Datentyps oder des Typs `void` wird mit ????? symbolisiert. Funktionen besitzen keinen Wert.

Bei einfachen Datentypen und Unterbereichstypen wird der Wert in Dezimal- und Hexadezimal-Darstellung gezeigt, wobei die Ziffernzahl der Hex-Darstellung von der Breite der Daten in Bits abhängt. Wenn der Wert der Zahl klein genug ist, wird sie zusätzlich als Zeichen gezeigt, dabei finden auch die Escape-Sequenzen der aktuellen Quellsprache Verwendung. Bei Aufzählungstypen wird nach Möglichkeit der (erste) Name, der für diesen Wert und diesen Typ definiert wurde, gezeigt.

Fließkommavariablen werden als Fließkommazahl angezeigt, dabei hängt die Anzahl der verwendeten Ziffern vom Datentyp ab.

Zeiger und Referenzen werden in Hex gezeigt, und nach einem Pfeil folgen noch die Daten, auf die der Zeiger oder die Referenz zeigt.

Die Werte der Elemente eines Feldes, einer Klasse oder eines Verbundes werden in geschweiften Klammern aufgezählt, jeweils durch ein Komma getrennt. Falls diese Elemente wiederum Felder, Klassen oder Verbunde sind, werden deren Elemente in geschweiften Klammern geschachtelt aufgezählt, aber darin erscheinen weitere Felder, Klassen und Verbunde nur noch als „...".

4. Die Menüs

In diesem Abschnitt finden Sie eine Auflistung aller Menüs des MaxonSDB und eine kurze Erklärung zu jedem Menüpunkt.

Aufbau der Menüs

Um die Benutzung von MaxonSDB zu vereinfachen, besitzt jedes Fenster das gleiche, global gültige Menü. Da aber fast jedes Fenster noch Befehle zur Verfügung stellt, die in anderen Fenstern (so) nicht vorhanden sind, wird für diese Fenster der globale Menübaum um ein lokales Menü erweitert.

Sie müssen nicht wie bei anderen Debuggern obskure Tastendrücke durchführen, um an dieses lokale Menü zu gelangen — es ist immer sofort mit der Maus erreichbar, und fast alle Befehle darin besitzen auch ein Tastaturkürzel.

Alle Menübefehle

Da MaxonSDB ja zweisprachig ist, wird bei jedem Menüpunkt sowohl der deutsche als auch der englische Menütext mit angegeben. Das Tastaturkürzel ist in allen Sprachen gleich, damit ein Sprachenwechsel keine Verwirrung auslöst.

Das Projekt-Menü

Englisch: „Project"
Dieses Menü enthält Befehle, um ein zu debuggendes Programm zu laden, aktuelle Informationen über MaxonSDB anzuzeigen, kontextsensitive Hilfe anzufordern und den Debugger zu verlassen.

Programm laden...

Englisch: „Load Program..."
Kürzel: <Amiga O>
Lädt ein Programm in den Speicher und bereitet es zum Debuggen vor. Falls derzeit noch ein anderes Programm debuggt wird, muß dieses erst zuende ausgeführt werden, damit

ein neues Programm geladen werden kann. Mit diesem Befehl werden automatisch auch alle Haltepunkte gelöscht. Falls „Zeige Quelltext" angewählt ist, wird das Programm schon mal bis zum Start der `main()`-Funktion ausgeführt.

Info/Über...

Englisch: „Info/About..."
Hiermit wird ein Fenster geöffnet, in dem (grundlegende und aktuelle) allgemeine Informationen über MaxonSDB zu sehen sind.

Hilfe...

Englisch: „Help..."
Kürzel: <Help>
Falls Sie das geniale interaktive Hilfesystem *HotHelp* installiert haben, wird dieses aufgerufen, und Sie erhalten kontextsensitive Hilfe zu MaxonSDB.

Debugger beenden

Englisch: „Quit Debugger"
Kürzel: <Amiga Q>
Irgendwann kommt immer der Zeitpunkt, an dem alle Fehler gefunden sind, und dann möchte man den Debugger wieder verlassen. Da ein laufendes Programm bei diesem Befehl einfach abgebrochen wird, und es somit keine Chance hätte, alle seine Ressourcen (Speicher, Fenster ect.) wieder freizugeben, erscheint vorher ein Dialogfenster, der Sie darauf aufmerksam macht, daß Sie gerade dabei sind, MaxonSDB zu verlassen. Dieses Dialogfenster erscheint auch dann, falls Sie das letzte offene Fenster des MaxonSDB schließen wollen, da Sie diesem ja danach keine Befehle mehr geben könnten.

Das Zeige-Menü

Englisch: „View"
Mit den Befehlen in diesem Menü können Sie verschiedene Fenster des MaxonSDB öffnen, in denen dann Informationen über das zu debuggende Programm dargestellt werden.

Stapel

Englisch: „Stack“
Kürzel: <Alt S>
Hiermit öffnen Sie das Fenster, in dem immer die aktuelle Aufrufkette der Funktionen des Programms dargestellt wird.

Aktuelle Var.

Englisch: „Current Locals“
Kürzel: <Alt V>
Dieser Befehl öffnet das Fenster, in dem immer die Funktionsargumente und lokalen Variablen der aktuellen Funktion dargestellt werden. Falls die aktuelle Funktion wechselt, werden in diesem Fenster andere Variablen gezeigt.

Globale Var.

Englisch: „Globals“
Kürzel: <Alt G>
Falls genau ein Modul geladen ist oder Sie im Modul-Fenster bereits ein Modul angewählt haben, werden alle globalen Variablen dieses Moduls in einem neuen Fenster angezeigt. Andernfalls werden Sie aufgefordert, zuvor ein Modul anzuwählen.

Funktionen

Englisch: „Functions“
Kürzel: <Alt F>
Analog zu obigem Menüpunkt werden hiermit die globalen Funktionen des angewählten Moduls angezeigt.

Überwachen

Englisch: „Watches“
Kürzel: <Alt W>
In diesem Fenster werden nur die Variablen gezeigt, die Sie explizit dort abgelegt haben. Falls Sie sich z.B. nur für die Werte von wenigen globalen Variablen aus verschiedenen Modulen interessieren, können Sie diese in das Überwachen-Fenster eintragen, und behalten dadurch leicht den Überblick.

Module

Englisch: „Modules“
Dieses Fenster zeigt Ihnen die Liste der geladenen Module, alphabetisch sortiert.

Speicher

Englisch: „Memory“
Kürzel: <Alt M>
In diesem Fenster wird der Inhalt des Hauptspeichers auf verschiedene, wählbare Arten gezeigt.

CPU-Register

Englisch: „CPU-Registers“
Kürzel: <Alt C>
In diesem Fenster werden die Prozessorregister und -Flags und die Adresse der Prozeßstruktur gezeigt.

Das Bearbeite-Menü

Englisch: „Edit“
Damit Sie sich nicht an einen neuen Menüaufbau in späteren Versionen von MaxonSDB gewöhnen müssen, wurde für die untenstehenden Menüpunkte gleich ein ganzes Menü spendiert.

Zeige Zeile...

Englisch: „Show Line...“
Kürzel: <^G>
Öffnet das Texteingabe-Fenster, in dem Sie die im Hauptfenster zu zeigende Zeilennummer eingeben können.

Zeige PC

Englisch: „Show PC"
Kürzel: <^P>
Zeigt im Hauptfenster die aktuelle Umgebung des Programmzählers, entweder im Quelltext, disassembliert oder beides.

Schritt

Englisch: „Step"
Mit den Befehlen dieses Menüs kann das Programm in kleineren oder größeren Schritten abgearbeitet werden, und Sie können die Programmargumente angeben.

Gehe

Englisch: „Go"
Kürzel: <F9> oder <^F9> oder <Alt F9>
Hiermit wird das Programm mit der normalen Geschwindigkeit ausgeführt, bis es auf einen Haltepunkt trifft,endet oder von Ihnen unterbrochen wird.

Folge hinein

Englisch: „Trace into"
Kürzel: <F7>
Dieser Befehl führt einen einzelnen Schritt aus, also einen Maschinenbefehl oder eine Quelltextzeile, je nachdem, ob im Hauptfenster Assembler gezeigt wird oder nicht.

Schritt über

Englisch: „Step over"
Kürzel: <F8>
Dies ist fast dasselbe wie „Folge hinein", aber Funktionsaufrufe werden am Stück mit normaler Geschwindigkeit ausgeführt. Verwenden Sie diesen Befehl, wenn die in der aktuellen Zeile stehende Funktion fehlerfrei ist, und Sie sie nicht mit einzelnen Schritten abarbeiten wollen.

Gehe bis Rückkehr

Englisch: „Go Until Return"
Kürzel: <Alt F8>
Falls Sie versehentlich (z.B. mit „Folge hinein") in eine Funktion hineingeraten sind, die Sie nicht interessiert, oder falls Sie sicher sind, daß der Rest der aktuellen Funktion keinen Fehler mehr enthält, können Sie hiermit die Funktion mit fast normaler Geschwindigkeit bis zum Ende ausführen, MaxonSDB hält Ihr Programm am Ende der Funktion automatisch an.

Gehe ohne Haltepkte

Englisch: „Go without Bkpts"
Falls Sie das Programm zuende ausführen wollen, ohne daß es noch einmal unterbrochen wird, benutzten Sie diesen Befehl. Er bewirkt fast das gleiche wie „Gehe", aber die Haltepunkte werden nicht aktiviert. Sie werden aber nicht gelöscht, d.h. sie sind nach Beendigung dieses Befehls noch vorhanden. Wenn Sie während des weiteren Programmlaufs neue Haltepunkte setzen, werden diese (trotzdem) sofort aktiviert.

Stopp

Englisch: „Stop"
Kürzel: <Amiga .>
Dies ist eine Spezialität des MaxonSDB, die Sie bei fast keinem anderen Debugger auf dem Amiga finden können!

Falls Ihr Programm in eine Endlosschleife geraten sollte, oder falls es ewig erfolglos wartet (`Wait()`), kann es hiermit unterbrochen werden und Sie können untersuchen, was los ist. Sie sollten aber in diesem Fall damit rechnen, daß das Programm sich gerade im ROM oder an sonstigen Plätzen aufhält, zu denen MaxonSDB den Quelltext nicht kennt. Falls Ihnen die dann angezeigte Maschinensprache nicht weiterhilft, probieren Sie (eventuell mehrfach) mit dem Befehl „Gehe bis Rückkehr" in den Bereich Ihres Programms zurückzukehren, für den Quelltext existiert und der diese Maschinensprachen-Unterprogramme aufgerufen hat.

Wenn Ihr Programm gerade wartet, wenn Sie den Befehl zum Stoppen geben, dann erscheint ein Dialogfenster, der Sie darauf aufmerksam macht, daß Ihr Programm zwar ohne weiteres angehalten werden kann, Ihnen dies aber vielleicht nicht weiterhilft. Wenn Sie in diesem Dialogfenster nicht „Zurück" sondern „Ich bin ein Guru!" wählen und dann Ihr Programm weiter abarbeiten, kann es sein, daß der Rechner abstürzt. Genauere Informationen finden Sie bei der Beschreibung dieses Dialogfensters.

Falls sich eine Endlosschleife innerhalb eines durch `Forbid()` oder `Disable()` geschützten Bereichs befindet, kann MaxonSDB Ihr Programm nicht auf diese Art unterbrechen, denn er erfährt in diesem Fall nichts von Ihrem Tastendruck, und Menüs sind dann ja auch nicht aufrufbar.

Neu Starten

Englisch: „Program Reset"
Kürzel: <^F2>
Dies ist fast dasselbe wie „Programm laden...", nur daß Sie nicht mehr nach dem Namen des Programms gefragt werden, da MaxonSDB diesen ja schon kennt. Auch hier werden alle Haltepunkte gelöscht.

Argumente...

Englisch: „Arguments..."
Hiermit können Sie die Kommandozeilenargumente Ihres Programms eingeben oder verändern. Dabei werden die neuen Argumente erst beim nächsten (Neu-) Start an ihr Programm übergeben, da es diese geänderten Argumente entweder sowieso ignorieren oder dadurch gründlich irritiert werden würde.

Wenn das Programm soeben frisch geladen wurde, und der Debugger dabei **nicht** als „Assemblerdebugger" konfiguriert war, sind die Kommandozeilenargumente Ihres Programms bereits ausgewertet, und dieser Befehl kommt zu spät.

Das Haltepunkte-Menü

Englisch: „Breakpoints"
Alle Befehle in diesem Menü beziehen sich auf die Verwaltung von Haltepunkten, wobei es derzeit nur einen Befehl gibt.

Alle löschen

Englisch: „Delete All"
Alle im Programm verstreuten Haltepunkte werden gelöscht, auch wenn das Programm gerade laufen sollte.

Das Konfigurations-Menü

Englisch: „Settings"
Mit den Befehlen dieses Menüs können Sie die Konfiguration von MaxonSDB einstellen und abspeichern oder eine andere Konfiguration laden.

Allgemein...

Englisch: „General..."
Dieser Befehl öffnet einen Dialogfenster, in dem Sie allgemeine Einstellungen für die Arbeit mit MaxonSDB treffen können. Siehe dazu das Kapitel „Allgemeine Parameter".

Lade Konfiguration...

Englisch: „Load Settings..."
Damit Sie nicht bei jedem Start von MaxonSDB alle Einstellungen neu setzen müssen, kann MaxonSDB sie abspeichern und wieder laden. Mit diesem Befehl können Sie sich aussuchen, welche von vielleicht mehreren Dateien mit Einstellungen Sie als aktuelle Einstellungen wünschen. Beim Start von MaxonSDB wird automatisch die Datei `MaxonSDB.prefs` geladen, wobei er diese bei Bedarf in mehreren Verzeichnissen sucht. Mehr darüber finden Sie im Kapitel „Konfiguration".

Sichere Konfig.

Englisch: „Save Settings"
Hiermit werden die aktuellen Einstellungen in die zuletzt gewählte Datei mit Einstellungen abgespeichert. Falls bisher noch keine solche Datei gewählt wurde, werden die Einstellungen in die Datei `MaxonSDB.prefs` gesichert, wobei das Verzeichnis davon abhängt, wo MaxonSDB bei seinem Start eine solche Datei gefunden hat. Falls er beim Start eine solche Datei gefunden hat, so wird diese überschrieben. Wenn die Datei in einen anderes Verzeichnis geschrieben werden soll, nehmen Sie dafür den Befehl „Sichere Konf. als...".

Sichere Konf. als...

Englisch: „Save Settings As..."
Dieser Befehl gibt Ihnen eine genaue Kontrolle darüber, in welche Datei in welchem Verzeichnis MaxonSDB die Konfiguration schreibt.

Das Fenster-Menü

Englisch: „Windows"
Da es bei MaxonSDB sehr wichtig ist, schnell zwischen mehreren Fenstern wechseln zu können, auch wenn man gerade die Finger auf der Tastatur hat, ist dieses Menü recht flexibel und umfangreich.

Zoom

Englisch: „Zoom"
Kürzel: <F5>
Ab Kickstart 2.0 kann hiermit ein Fenster verkleinert und wieder auf Originalgröße gebracht werden.

Aktuell

Englisch: „Update All"
Kürzel: <Alt U>
Falls sich die Informationen im Speicher schneller ändern, als MaxonSDB annimmt, können Sie ihn mit diesem Befehl dazu bringen, alle Daten erneut einzulesen und darzustellen.

Neuzeichnen

Englisch: „Redraw All"
Kürzel: <Alt R>
Weil es durchaus vorkommen kann, daß Ihr Programm wild auf den Bildschirm schreibt und dabei Fenster von MaxonSDB übermalt, können Sie MaxonSDB hiermit dazu bringen, alle Fenster komplett neu zu malen.
Hinweis: Sie sollten diesen Befehl normalerweise nicht benötigen. Falls doch, dann ist entweder Ihr Programm fehlerhaft oder es wird bei späteren Versionen der Kickstart nicht unbedingt funktionieren!

Schließen

Englisch: „Close“
Kürzel: <Alt F3>
Hiermit können Sie das aktuelle Fenster auch über die Tastatur schließen.
Hinweis: Mit <Shift F3> werden alle Inspect-Fenster auf einmal geschlossen. Hierfür gibt es aber keinen Befehl im Menü!

Dieses

Englisch: „This one“
Kürzel: <Amiga 0>
Bringt das aktuelle Fenster nach vorne, wenn es von anderen Fenstern verdeckt sein sollte.

Nächstes

Englisch: „Next“
Kürzel: <Amiga +> oder <F6>
Wechselt das aktuelle Fenster, so daß das Fenster mit der nächsthöheren Nummer zum aktuellen Fenster wird. Falls es kein Fenster mit einer höheren Nummer gibt, dann wird das Fenster mit der niedrigsten Nummer zum aktuellen Fenster.

Vorheriges

Englisch: „Previous“
Kürzel: <Amiga -> oder <Shift F6>
Wechselt das aktuelle Fenster, so daß das Fenster mit der nächstniedrigeren Nummer zum aktuellen Fenster wird. Falls es kein Fenster mit einer niedrigeren Nummer gibt, dann wird das Fenster mit der höchsten Nummer zum aktuellen Fenster.

Fenster 1 bis Fenster 10

Englisch: „Window #1“ bis „Window #10“
Kürzel: <Alt 1> bis <Alt 9> und <Alt 0>
Falls Sie im Kopf haben, welches Fenster welche Nummer hat, können Sie mit diesen zehn Kommandos sehr schnell und exakt das gewünschte Fenster aktivieren. Falls mehr

als zehn Fenster von MaxonSDB auf dem Bildschirm sind, so können Sie auf einem Standard-Amiga sowieso nicht mehr alle Informationen gleichzeitig sehen und sollten dann Fenster schließen, oder sich mit den Befehlen „Vorheriges" und „Nächstes" durchhangeln. Daß der Rechner bei mehr als 20 Fenstern unerträglich langsam wird, ist kein Problem speziell von MaxonSDB, sondern liegt an der Layers.library, die mit ineffizienten Algorithmen arbeitet. (Die Dos.library hat Commodore ja mit Kickstart 2.0 erheblich beschleunigt, meiner Meinung nach sollte jetzt bei der nächsten Kickstart mal gründlich in der Layers.library aufgeräumt werden.)

Die lokalen Menüs

Englisch: „Local"
Die große Flexibilität von MaxonSDB liegt in den lokalen Menüs, auf deren Befehle jedes Fenster auf seine eigene Art reagiert und dabei andere Fenster beeinflußt. Das Modul-Fenster z.B. zeigt beim Befehl „Quelltext" die Haupt-Datei des in ihm aktuell gewählten Moduls im Hauptfenster, und ein Klassen-Inspect-Fenster zeigt dort die Deklaration der in ihm angezeigten Klasse.

Um dabei beim Benutzer keine Verwirrung zu erzeugen, gibt es nur eine kleine, sorgfältig ausgewählte Menge an Befehlen, die in lokalen Menüs auftreten. Da jedes Fenster auf diese Befehle anders reagiert, finden Sie hier nur eine kurze, allgemeine Beschreibung zu jedem Befehl, und speziellere Informationen bei der Beschreibung der einzelnen Fenster.

Quelltext

Englisch: „Show Source"
Kürzel: <F3>
Zeigt den Quelltext zum angewählten Objekt im Hauptfenster. Wenn das Hauptfenster derzeit geschlossen ist, wird es durch diesen Befehl geöffnet.

Überwachen

Englisch: „Watch"
Kürzel: <^F7>
Zeigt das angewählte Objekt (zusätzlich) im Überwachen-Fenster. Das Überwachen-Fenster muß für diesen Befehl nicht geöffnet sein, und es wird auch nicht automatisch geöffnet. (Sie öffnen das Überwachen-Fenster mit dem Befehl „Zeige/Überwachen".)

Inspizieren

Englisch: „Inspect"
Kürzel: <^I>
Öffne ein Inspect-Fenster zu dem ausgewählten Objekt. Inspect-Fenster lassen sich **nur** mit diesem Befehl öffnen, es gibt keinen entsprechenden Befehl im „Zeige"-Menü.

Absteigen

Englisch: „Descend"
Kürzel: <^D>
Dies entspricht einem „Inspizieren"-Befehl mit anschließendem „Schließen"-Befehl für das vorher aktuelle Fenster, wobei die Fensternummer und -Position beibehalten wird. Dieser Befehl ist nützlich, falls Sie z.B. eine verkettete Liste bis zu einem bestimmten Knoten verfolgen wollen, und die vorherigen Knoten Sie nicht weiter interessieren.

Lokale Var.

Englisch: „Local Vars"
Kürzel: <^V>
Zeige die lokalen Variablen der aktuell angewählten oder gezeigten Funktion, mit ihren Werten, falls diese verfügbar sind.

Speicher

Englisch: „Memory"
Kürzel: <^M>
Öffnet ein Speicherfenster ab der Adresse des gewählten oder gezeigten Objekts.

Löschen

Englisch: „Delete"
Kürzel: <Del>
Das angewählte Objekt wird aus dem Fenster entfernt und eventuell gelöscht.

Alles Löschen

Englisch: „Delete All“
Alle in diesem Fenster dargestellten Objekte werden gelöscht.

Neue Adresse...

Englisch: „New Address...“
Kürzel: <^A>
Das Texteingabe-Fenster öffnet sich, in dem Sie eine Adresse angeben können, ab der dann die Daten im Fenster gezeigt werden.

Indexbereich...

Englisch: „Index Range...“
Kürzel: <^R>
Das Texteingabe-Fenster öffnet sich, in dem Sie den ersten und letzten Index eingeben können, der Sie bei interessiert.

Adressgrenzen...

Englisch: „Address Range...“
Kürzel: <^R>
Das Texteingabe-Fenster öffnet sich, in dem Sie die erste und letzte gültige Adresse eingeben können.

Daten ändern...

Englisch: „Change...“
Kürzel: <^C>
Es ist keineswegs selbstverständlich, daß ein Debugger erlaubt, daß die Werte der angezeigten Variablen von Hand geändert werden können. Nun, MaxonSDB kann dies natürlich. Dieser Befehl öffnet das Texteingabe-Fenster, in dem Sie dann einen neuen Wert für das angewählte Objekt eingeben können. Die Daten werden (wie immer) erst dann und nur dann geändert, wenn Sie das Texteingabe-Fenster mit „OK“ verlassen.

als Byte
als Word
als Long

Englisch: „As Bytes As Words As Longs"
Kürzel: <^1> und <^2> und <^4>
Hiermit können Sie wählen, wie die Daten im Speicher zusammengefaßt werden, bevor sie ausgegeben werden.

Dezimal
Hexadezimal
Oktal Basis 2
7-Bit-ASCII
UniCode

Englisch: „Decimal" „Hexadecimal" „Octal" „Base 2" „7 Bit ASCII" „UniCode"
Kürzel: <^D> <^H> <^O> <^B> <^7> <^8>
Hiermit können Sie wählen, wie die Daten ausgegeben werden, nachdem sie aus dem Speicher (oder Register) gelesen wurden. 7-Bit-ASCII gibt für die Zahlen 32 bis 126 das entsprechende ASCII-Zeichen aus und für alle anderen Zahlenwerte den Punkt („."). UniCode nimmt die komplette Zahl (im Wert von 0 bis $7ffffffe) und gibt das entsprechende UniCode-Zeichen aus. Derzeit wird aber nur für die Zahlen 32 bis 126 und 160 bis 255 das UniCode-Zeichen ausgegeben, alle anderen Zeichen erscheinen als Punkt.

Sortieren aus
nach Name
nach Ort

Englisch: „Do not Sort Sort by Name by Address"
Kürzel: <^0> und <^1> und <^2>
In manchen Fenstern kann man die Reihenfolge der angezeigten Objekte beeinflussen. „Sortieren aus?" zeigt die Objekte in der Reihenfolge, wie sie in der `#?.mdbi`-Datei abgelegt sind, dies sollte in etwa der Reihenfolge entsprechen, wie der Compiler sie in den Quelldateien gefunden hat. „Sortieren nach Name" sortiert die Objekte anhand ihrer Namen alphabetisch aufsteigend, wobei Groß- und Kleinbuchstaben nicht unterschieden werden. „Sortieren nach Ort" schließlich ordnet die Objekte nach aufsteigenden Speicheradressen.

Zeige Quelltext
Zeige Assembler
Zeige beides

Englisch: „Show Source Show Assembly Show Both“
Kürzel: <^1> <^2> <^3>
Falls der erste Menüpunkt angewählt ist, wird MaxonSDB nach Möglichkeit den Quelltext, der zum aktuellen Wert des Programmzählers gehört, zeigen und nur in Ausnahmefällen diese Stelle disassembliert zeigen. Auch werden bei Einzelschritten dann komplette Quelltextzeilen abgearbeitet.

Wenn aber der zweite oder dritte Menüpunkt angewählt ist, wird MaxonSDB die Umgebung des aktuellen Programmzählers immer als Disassemblerlisting zeigen, und bei Einzelschritten wird nur ein Maschinenbefehl ausgeführt. Beim dritten Menüpunkt wird zusätzlich noch der Quelltext zu den Maschinenbefehlen gezeigt.

5. Haltepunkte

Haltepunkte sind dazu da, das Debuggen zu beschleunigen. Anstatt eine Programmzeile nach der anderen im Einzelschrittmodus abzuarbeiten, bis das Programm endlich die gewünschte Zeile erreicht hat, setzen Sie einfach einen Haltepunkt in dieser Zeile und lassen das Programm laufen. MaxonSDB hält Ihr Programm an, sobald es diese Zeile erreicht hat.

Sie können einen Haltepunkt jederzeit setzen, löschen oder ändern, unabhängig davon, ob Ihr Programm gerade läuft oder nicht.

Haltepunke setzen

Einen Haltepunkt setzen Sie ganz einfach dadurch, daß Sie im Hauptfenster in der gewünschten Zeile ganz links das Achteck anklicken. Wenn eine Zeile kein solches Achteck hat, kann auch kein Haltepunkt dort gesetzt werden. Setzen Sie in diesem Fall den Haltepunkt auf die Zeile davor oder danach, die ein solches Achteck besitzt.

Wenn Sie einen Haltepunkt in eine Zeile setzen, zeigt diese ein gefülltes Achteck in einer auffälligeren Farbe, so daß Sie jederzeit diesen Haltepunkt finden können.

Durch erneutes Anklicken können Sie den Haltepunkt wieder löschen.

Symbole im Hauptfenster

Im Hauptfenster befindet sich links des angezeigten Textes noch ein senkrechter Streifen, in dem ein oder zwei Symbole zu jeder Zeile gezeigt werden. Diese Symbole werden hier erklärt.

Die Symbole sagen Ihnen, ob Sie in dieser Zeile einen Haltepunkt setzen können oder schon einer gesetzt ist, und ob der Programmzähler sich in dieser Zeile befindet. Haltepunkte werden dabei durch verschiedenfarbige Achtecke und der Programmzähler wird durch einen Pfeil symbolisiert.

Falls der Compiler für eine Zeile keinen Code erzeugte, ist die zugehörige Symbolfläche immer leer, denn dann kann hier kein Haltepunkt gesetzt werden.

Symbolisierung des Programmzählers

Wenn der Programmzähler sich in einer Zeile befindet, erscheint (immer über andere, bereits dort vorhandene Symbole gemalt) ein Pfeil in der Farbe SHINEPEN, dies ist üblicherweise helles Weiß und der Pfeil somit sehr auffällig.

Falls der Programmzähler nicht auf den ersten Maschinenbefehl einer Zeile zeigt, dann wird der Pfeil nicht flächig sondern hohl dargestellt. Dies kann z.B. dann vorkommen, wenn eine Zeile mehrere Funktionsaufrufe enthält, und das Programm gerade von der ersten Funktion zurückgekehrt ist und die nächste Funktion noch nicht begonnen wurde.

Symbolisierung der Haltepunkte

Wenn Sie in einer Zeile einen Haltepunkt setzen können, finden Sie dort ein hohles Achteck in der Farbe TEXTPEN, dies ist unter Kickstart 2.0 normalerweise schwarz (auf grauem Hintergrund). Dieses Achteck soll ein STOP-Schild darstellen, da das Programm an dieser Stelle angehalten werden kann. Möglicherweise sehen Sie aber auch statt eines echten Achtecks nur einen Kreis, das kommt ganz darauf an, was Sie dort sehen wollen...

Wenn Sie dieses Achteck mit der Maus anklicken, erscheint ein recht auffälliges ausgefülltes Achteck in der Farbe FILLPEN (normalerweise blau, unter Kickstart 1.3 orange). Dies symbolisiert einen gesetzten Haltepunkt.

Wenn Sie einen einfachen Haltepunkt nochmal mit der Maus anklicken, verschwindet er wieder, und es erscheint wieder das schwarze hohle Achteck.

Wenn Sie einen Doppelklick auf ein (hohles oder gefülltes) Achteck anwenden, öffnet sich das Haltepunkt-Dialogfenster, und Sie können Bedingungen zu diesem Haltepunkt angeben. Wenn Sie Bedingungen zu einem Haltepunkt angegeben haben, ist dieser kein einfacher Haltepunkt mehr, und er kann dann auch nicht mehr mit einem Mausklick gelöscht werden. Er wird statt dessen durch einen Mausklick inaktiviert. Ein solcher inaktiver Haltepunkt wird durch ein auffälliges hohles Rechteck (ebenfalls in der Farbe FILLPEN) dargestellt. Ein weiterer Mausklick aktiviert einen inaktiven Haltepunkt wieder, und ein Doppelklick öffnet (wie immer) dasHaltepunkt-Dialogfenster.

Wenn Sie einen komplexen Haltepunkt löschen wollen, so können Sie dies nur im Haltepunkt-Dialogfenster tun, siehe unten. Oder Sie wählen den Menübefehl „Haltepunkte/Alle löschen“, aber dann werden alle Haltepunkte gelöscht.

Bedingte Haltepunkte

Oft reicht es nicht aus, ein Programm jedesmal anzuhalten, wenn eine bestimmte Zeile erreicht wurde. Dies ist insbesondere dann der Fall, wenn eine Zeile sehr oft ausgeführt wird, bevor man überhaupt an dieser Zeile Interesse hat.

Hierfür kann man in MaxonSDB einen Haltepunkt mit einer Bedingung und/oder einem Zähler ausstatten, so daß das Programm nur dann anhält, wenn die Bedingung soundso oft zugetroffen hat. Diese zusätzlichen Eigenschaften eines Haltepunktes setzen Sie im Haltepunkt-Dialogfenster.

Wenn Sie das Programm anhalten wollen, wenn eine Funktion zum zehnten Mal ausgeführt wird, dann können Sie einen Haltepunkt an den Anfang des Funktionsrumpfs setzen und diesen Haltepunkt mit einem Zähler versehen. Dieser Zähler erhöht seinen Wert jedesmal, wenn der Haltepunkt erreicht wird, und bei Erreichen der eingestellten Schranke wird das Programm angehalten.

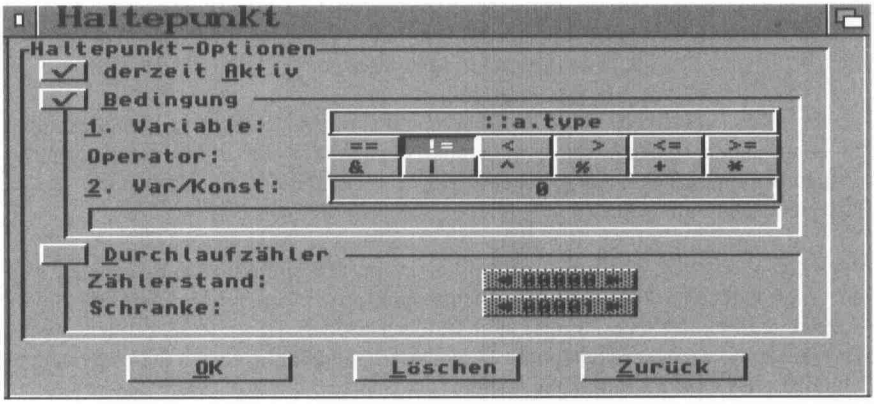
Wenn Sie das Programm anhalten wollen, wenn eine Funktion mit dem Argument 42 aufgerufen wird, dann können Sie einen Haltepunkt an den Anfang des Schleifenrumpfs setzen und statten diesen Haltepunkt mit der Bedingung `x == 42` aus (wenn das Argument `x` heißt). Jetzt wird jedesmal beim Erreichen des Haltepunktes geprüft, ob die Bedingung zutrifft, und gegebenenfalls wird das Programm angehalten.

Wenn Sie das Programm schließlich anhalten wollen, wenn eine Funktion zum zehnten mal mit dem Argument 42 aufgerufen wird, dann statten Sie den obigen Haltepunkt sowohl mit dem Zähler, als auch mit der Bedingung aus. Jetzt wird jedesmal beim Erreichen des Haltepunktes die Bedingung geprüft, und falls diese zutrifft, wird der Zähler erhöht. Wenn dieser die Schranke erreicht, wird das Programm angehalten.

Das Haltepunkt-Dialogfenster

Wenn Sie im Hauptfenster das zu einer Zeile gehörende Achteck zweimal schnell anklicken (Doppelklick), öffnet sich dasHaltepunkt-Dialogfenster. In diesem Dialogfenster können Sie sowohl eine Bedingung als auch einen Zähler zu diesem Haltepunkt angeben. Wenn Sie keine Bedingung angeben, so verhält sich der Haltepunkt genau so, wie wenn die Bedingung immer zuträfe. Wenn Sie den Zähler nicht angeben, dann wird bei jedem Erreichen des Haltepunktes das Programm angehalten.

In diesem Dialogfenster können Sie folgende Einstellungen vornehmen:



Bei **derzeit aktiv** können Sie wählen, ob der Haltepunkt tatsächlich aktiv ist oder derzeit ruht. Ein ruhender Haltepunkt wird bei der Ausführung des Programms vollständig ignoriert. Diese Möglichkeit ist sinnvoll, wenn Sie einen komplizierten Haltepunkt definiert haben, den Sie im Moment nicht brauchen. Sie können den Aktivitätszustand eines komplexen Haltepunkts auch durch einfaches Anklicken seines Symbols im Hauptfenster ändern. Einfache Haltepunkte werden durch Anklicken gelöscht, komplexe Haltepunkte werden nur inaktiv.

Die **Bedingung**, die zu einem Haltepunkt angegeben werden kann, hat ein festes Format. Es ist ein Ausdruck mit zwei Variablen oder Konstanten und einem Operator in der Mitte. Die Variablen werden an der Stelle ausgewertet, an der der Haltepunkt steht. Es ist also nicht möglich, hier auf die Variablen einer anderen Funktion zuzugreifen, und bei Rekursion beziehen sich die Namen lokaler Variablen immer auf die Variablen der aktuellen Funktion. Falls es sowohl eine lokale als auch eine globale Variable mit dem Namen `x` gibt, so können Sie die globale Variable unter dem Namen `::x` ansprechen. Die statischen Klassenelemente werden wie üblich mit „Klassenname::Element" angesprochen. Sie können hier nur einfache, nicht zusammengesetzte Variablen angeben.

Als Operatoren stehen die sechs Vergleichsoperatoren und noch ein paar andere zur Verfügung. Nicht vorhandene Operatoren lassen sich mit den angegebenen nachbilden, so ist z.B. `if (x - y)` dasselbe wie if `(x != y)`. Die Werte der Variablen oder Konstanten werden immer zuerst nach `long`, `unsigned long` oder `double` gewandelt (auch bei Zeichen und Zeigern!), bevor sie mit dem Operator verknüpft werden. Normalerweise ist `long` dasselbe wie `int`, aber falls Sie 16-Bit-`int`s benutzen, müssen Sie hier etwas achtgeben.

Wenn bei der Verknüpfung 0 oder 0.0 herauskommt, dann ist die Bedingung nicht erfüllt, und das Programm wird weiter ausgeführt.

Der **Durchlaufzähler** kann ebenfalls getrennt zugeschaltet werden. Er besteht aus einem Zähler und einer Schranke. Der Zähler startet normalerweise bei Null, und wird jedesmal, wenn bei Erreichen des Haltepunktes die Bedingung zutrifft (oder diese ganz fehlt), erhöht. Wenn der Zähler den Wert in der Schranke erreicht, wird das Programm angehalten und der Zähler auf Null gesetzt. Wenn bei jedem dritten Durchlauf das Programm angehalten werden soll, dann setzen Sie dazu die Schranke auf drei und löschen den Zähler. Wenn das Programm bei jedem dritten Durchlauf anhalten soll, aber schon zwei Durchläufe vorbei sind, bei denen der Haltepunkt noch nicht existierte, setzen Sie die Schranke auf drei und den Zähler auf zwei.

Der Wert in der Schranke kann nie kleiner oder gleich dem Wert im Zähler sein.

Wenn Sie einen komplexen Haltepunkt löschen wollen, so können Sie dazu das Löschen-Symbol unten im Dialogfenster verwenden. Sie können aber auch die Bedingung abschalten, den Zähler auf Null und die Schranke auf Eins setzen, so daß er wieder ein einfacher Haltepunkt wird, und diesen nach Beendigung des Dialogfensters mit einem Mausklick löschen.

6. Konfiguration

Es ist in gewissen Grenzen möglich, das Erscheinungsbild und Verhalten von MaxonSDB zu beeinflussen. Hierfür gibt es das Konfigurations-Menü, mit dem verschiedene Dialogfenster aufgerufen werden können, und auch mit Kommandozeilenoptionen können manche Optionen eingestellt werden. Alle diese Einstellungen, Lage, Größe und Aufteilung der Fenster können in einer Datei abgelegt und wieder eingelesen werden.

Die Datei MaxonSDB.prefs

Beim Start sucht MaxonSDB automatisch in mehreren Verzeichnissen nach einer Datei mit dem Namen MaxonSDB.prefs, aus der er die Konfiguration lesen kann. Falls er nirgendwo eine solche Datei findet, öffnet sich ein Dialogfenster, der Sie auf diesen Umstand aufmerksam macht. In diesem Fall gelten für alle Einstellungen die in MaxonSDB eingebauten Werte. Sie sollten dann alle Parameter nach Ihren Wünschen einstellen und diese dann in eine Datei namens MaxonSDB.prefs in einem der unten aufgeführten Verzeichnisse schreiben, so daß der Dialogfenster beim nächsten Aufruf von MaxonSDB nicht mehr erscheint.

Um eine gewisse Flexibilität auch in Netzwerken zu ermöglichen, sucht MaxonSDB diese Datei in mehreren Plätzen: Zuerst sucht er im aktuellen Verzeichnis, in dem er gestartet wurde. Falls er dort nicht fündig wird, sucht er weiter im Verzeichnis „S:", dann im Verzeichnis „MSDB:" und unter Kickstart 2.0 schließlich noch im Verzeichnis „PROGDIR:" (das ist das Verzeichnis, in dem das Programm „MaxonSDB" liegt). Falls er dabei nirgendwo fündig wird, nimmt er für alle Parameter die fest in ihm eingebauten Standardwerte.

MaxonSDB schreibt eine Menge Daten in solch eine Konfigurationsdatei: Dort werden natürlich alle im Konfigurations-Menü gewählten Einstellungen abgelegt. Zusätzlich steht dort Lage und Größe der Fenster (außer Inspect-Fenster) und deren Fensterkacheln. Bei manchen Fenstertypen wird auch noch abgelegt, ob diese gerade geöffnet sind, diese werden dann beim nächsten Programmstart automatisch geöffnet.

Da dies eine ganze Menge Informationen darstellt, wird die Datei in einem Binärformat abgelegt und ist deshalb nicht im Klartext lesbar.

Die Konfigurations-Dialogfenster

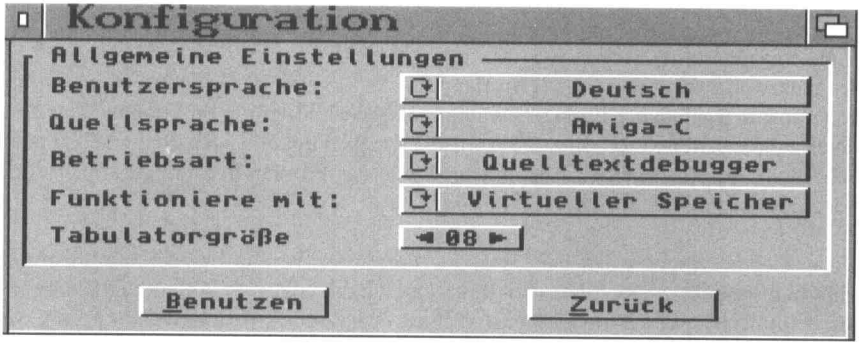
Es gibt mehrere Dialogfenster, in denen Sie das Verhalten von MaxonSDB und das Aussehen seiner Fenster einstellen können.

Mit dem Dialogfenster, der durch den Menübefehl „Konfig./Allgemein..." gezeigt wird, können Sie folgende Einstellungen treffen:

Allgemeine Parameter

In diesem Dialogfenster können Sie allgemeine Einstellungen für die Arbeit mit MaxonSDB setzen. Dazu gehören:

- Die Sprache, in der alle Texte ausgegeben werden.
- Die Programmiersprache, in der Typen und Werte ausgegeben werden.
- Ob das Programm automatisch gestartet werden soll oder nicht.
- Ob MaxonSDB den Enforcer oder virtuellen Speicher unterstützen soll.
- Den von Ihnen in den Quelltexten bevorzugten Tabulator.



Benutzersprache:

Englisch: „User Language:"
MaxonSDB ist (mit Ausnahme der Hilfstexte) vollständig zweisprachig. Dies bedeutet, daß alle Texte des MaxonSDB sowohl auf deutsch als auch auf englisch ausgegeben werden können. Wenn Sie die englischen Fachausdrücke bevorzugen, können Sie MaxonSDB so umschalten, daß er alle seine Texte in Englisch ausgibt. Diese Wahl können Sie auch mit der Kommandozeilenoption -l oder -L treffen.

Quellsprache:

Englisch: „Source Language:“
Hiermit können Sie wählen, auf welche Art MaxonSDB Datentypen und Werte darstellt, und wie er Ausdrücke auswertet. Gegenwärtig haben Sie die Wahl zwischen „ANSI-C“ und „Amiga-C“.

In „Amiga-C“ wird der z.B. der Typ `unsigned short` als `UWORD` ausgegeben, so wie er in `exec/types.h` umdefiniert wurde, und in „ANSI-C“ ganz normal als `unsigned short`.

Welche der beiden „C“-Varianten Sie wählen, ist reine Geschmackssache. Ich empfehle „Amiga-C“ für echte Amiga-Programme, die `exec/types.h` einbinden und die darin definierten Datentypen verwenden, und „ANSI-C“ für Programme, die ohne Änderung z.B. auch auf UNIX-Maschinen laufen würden. Ansonsten ist die Darstellung von Datentypen mit „Amiga-C“ kompakter.

Diese Wahl können Sie auch mit der Kommandozeilenoption -s oder -S treffen.

Betriebsart:

Englisch: „Operating Mode:“
Normalerweise zeigt MaxonSDB im Hauptfenster nur Quelltext und führt den Startup-Code eines neu geladenen Programms automatisch aus, so daß der Programmzähler dann gleich auf der ersten Zeile von `main()` (oder wie die Funktion auch immer heißt) steht. Dabei wird normalerweise die Kommandozeile des Programms zerlegt, kopiert, und globale Klassenobjekte werden initialisiert.

Wenn Sie dies nicht wünschen, weil Ihr Programm bereits bei der Initialisierung fehlerhaft arbeitet, wählen Sie hier statt „Quelltextdebugger“ „Assemblerdebugger“, und beim nächsten Laden oder Neustart Ihres Programmes wird dessen Startup-Code nicht automatisch ausgeführt, so daß Sie diesen schrittweise abarbeiten können.

Wenn Sie, nachdem Ihr Programm frisch gestartet wurde, wieder auf „Quelltextdebugger“ umschalten, führt der erste Einzelschritt das Programm genau bis zur ersten Initialisierungsroutine aus, so daß Sie diese im Quelltext debuggen können und sich nicht lange mit einem Disassemblerlisting plagen müssen.

Technischer Hinweis: Bei diesem Einzelschritt wird der Startup-Code im Trace-Modus abgearbeitet. Dies ist dann ein Problem, wenn der Startup-Code kurzzeitig in den Supervisor-Modus schaltet. Dies ist z.B. dann der Fall, wenn er die FPU zurücksetzen will (`FRESTORE 0`), damit das Task-Switching schneller ist. Ohne das Hilfsprogramm „SafeTrace“ ist dann ein Guru unvermeidlich! Siehe dazu im Anhang das Kapitel über „SafeTrace“.

Funktioniere mit:

Englisch: „Support for:"
Hier können Sie einstellen, ob MaxonSDB besser mit dem Programm „Enforcer" oder mit virtuellem Speicher zusammenarbeiten soll. Wenn Sie beides nicht benutzen, ist diese Einstellung egal. Es ist prinzipiell nicht möglich, virtuellen Speicher und Enforcer gleichzeitig zu benutzen.

Den technischen Hintergrund zu dieser Einstellungsmöglichkeit finden Sie im Anhang im Kapitel über „Enforcer".

Tabulatorgröße:

Englisch: „Tab Size:"
Die normalerweise im Quelltext verwendete Tabulatorgröße können Sie hiermit einstellen. Der von MaxonSDB erlaubte Bereich ist 1 bis 40 einschließlich. Kurzfristig (d.h. für die gerade gezeigte Datei) können Sie im Hauptfenster den Tabulator mit den Tasten <+> und <-> des Zahlenblocks einstellen, diese Einstellung geht jedoch bei jedem Dateiwechsel verloren.

Der Standardwert für den Tabulator ist 8, und wenn Sie in Ihrem Editor einen anderen Wert hierfür angeben, können Sie Ihre Quelltexte nicht mehr problemlos im CLI oder der Shell mit `TYPE` betrachten und mit `COPY TO prt:` drucken.

Dieser Wert kann auch mit der Kommandozeilenoption -t oder -T gesetzt werden.

7. Dialogfenster, Informations- und Fehlermeldungen

MaxonSDB gibt alle Meldungen in Dialogfenstern oder in der Statuszeile des Hauptfensters aus. Wenn eine solches Dialogfenster erscheint, so muß dieser erst zur Kenntnis genommen und beantwortet werden, bevor man mit MaxonSDB weiterarbeiten kann.

Die Meldungen werden mit der Funktion `R_FormAlert()` der `rct.library` dargestellt.

Meldungen mit Dialogfenster

Hier werden alle vorkommenden Meldungen alphabetisch aufgelistet und erklärt.

Benötigt residente Libraries!

Englisch: „Needs resident libraries!"
MaxonSDB kann keine Programme debuggen, die beim Programmstart automatisch Libraries öffnen lassen, da dies von Commodore nicht mehr unterstützt wird. Es handelt sich bei diesen Programmen meistens um alte BCPL-Programme, zu denen ohnehin kein Quelltext vorliegt.

Bitte zuerst ein Modul wählen oder laden!

Englisch: „Please choose or load a module first!"
Sie haben kein Programm geladen oder Ihr Programm besteht aus mehreren Modulen, als Sie einen der Befehle „Zeige/Funktionen" oder „Zeige/Globale Var." gegeben haben, so daß MaxonSDB nicht weiß, auf welches Modul sich der Befehl bezieht. Laden Sie ein zu debuggendes Programm oder wählen Sie im (jetzt automatisch geöffneten) Modul-Fenster eines der Module.

ChipMem-Speichermangel!

Englisch: „Out of chip - memory!"
Beim Start von MaxonSDB ist nicht genügend CHIP-Memory frei, um die Graphikdaten (Bildchen) abzulegen. Schließen Sie ein paar Bildschirme oder Fenster, und starten Sie MaxonSDB neu.

Datei hat falsches Format!

Englisch: „File contents are strange!"
Eine einzuladende Einstellungen-Datei hat ein falsches Format und kann nicht geladen werden. Wenn es sich wirklich um eine Einstellungen-Datei von MaxonSDB handelt, dann löschen Sie diese Datei, da sie defekt ist.

Datei nicht gefunden!

Englisch: „File not found!"
MaxonSDB konnte eine von Ihnen angegebene oder in einer #?.mdbg- oder #?.mdbi-Datei referenzierte Datei nicht finden. Es ist problematisch, Debugger- oder Quelltextdateien nach der Übersetzung in andere Verzeichnisse zu bewegen, da dann die Pfadverweise in den Debuggerdateien nicht mehr stimmen.

Fehlerhafte ".mdbg"Datei!

Englisch: „Error in ".mdbg " - file!"
Eine einzuladende #?.mdbg-Datei hat falsches Format. Wenn Sie die Datei mit einem Editor verändert haben, machen Sie die fehlerhafte Änderung rückgängig, oder übersetzen und binden Sie Ihr Programm neu.

'HotHelp' ist nicht installiert!

Englisch: „'HotHelp' not available"
Um die Hilfefunktion von MaxonSDB benutzen zu können, benötigen Sie das Programm 'HotHelp' von Maxon, und dieses muß auch installiert sein.

Ihr Programm wartet derzeit auf Signale.

Englisch: „Your program ist waiting for some signals."
(Der Text im Dialogfenster geht noch weiter.)

Es ist problemlos möglich, daß MaxonSDB ein laufendes Programm unterbricht, solange es im Zustand „TS_READY" ist, und es kann auch ohne Probleme wieder weiter ausgeführt werden. Wenn das Programm aber im Zustand „TS_WAITING" ist, wenn es unterbrochen werden soll, so „befindet" sich das Programm im ROM (in der Funktion `Wait()`).

Wenn MaxonSDB es dann unter seine Kontrolle bringt, kann erstens kein Quelltext zum Programmzähler gezeigt werden und zweitens ist es sehr wahrscheinlich, daß das Programm das ROM ohne äußeren Eingriff nicht wieder verlassen kann, da es über die Funktion `WaitPort()` in den Wartezustand gebracht wurde, und zum Verlassen dieser Funktion nicht nur ein Signal, sondern eine echte Message am Port angekommen sein muß.

Sie müssen sich mit den Interna von Exec und mit Assemblersprache **sehr** gut auskennen, wenn Sie das Programm dann trotzdem aus dem ROM führen wollen, um die die ROM-Routinen aufrufende Stelle in Ihrem Programm zu finden.

Nur ein kleiner Hinweis, wer ihn nicht 100% versteht, läßt besser die Finger von der Sache: Sie müssen an bestimmten Punkten in `WaitPort()` den Zustand der Prozessorregister manipulieren, und bei der Rückkehr aus `WaitPort()` kann d0 keinen gültigen Zeiger auf eine Message enthalten, so daß Sie zumindest ein sich oft daran anschließendes `Remove()` o.ä. überspringen müssen, sonst stürzt der Rechner ab. Auch wenn Ihnen ein `Disable()` (oder das entsprechende Makro!!) unter den Programmzähler kommt, wird es gefährlich.

Internal error

Englisch: „Internal error"
Diese Fehlermeldung sollte eigentlich nicht auftreten. Sie zeigt einen internen Fehler von MaxonSDB an. Bitte benachrichtigen Sie Maxon Computer, und geben Sie eine genaue Beschreibung der Umstände, die zu dieser Fehlermeldung geführt haben.

Ist keine IFF-Datei!

Englisch: „Not an IFF-file!"
Eine einzuladende `#?.mdbi`-Datei hat ein falsches Format. Übersetzen Sie Ihr Programm neu.

Ist keine MDBI-Datei!

Englisch: „Not a MDBI-file!"
Eine einzuladende `#?.mdbi`-Datei hat ein falsches oder zu komplexes Format. Übersetzen Sie Ihr Programm neu.

Kann Datei nicht erzeugen!

Englisch: „Could not create File!"
Beim Schreiben einer Datei trat ein Fehler auf. Schaffen Sie genug Platz auf dem Datenträger, indem Sie unnötige Dateien löschen, entfernen Sie den Schreibschutz des Datenträgers und den der zu überschreibenden Datei, oder geben Sie einen Dateinamen auf einem anderen Dateiträger an, in den die Datei geschrieben werden kann.

Keine ausführbare Datei!

Englisch: „Not an executable file!"
Das Programm, das Sie debuggen wollen, ist gar nicht ausführbar. Vielleicht ist es ein BASIC-, SHELL- oder AREXX-Programm?

Kein Programm geladen!

Englisch: „No program loaded!"
Sie wollten einen Einzelschritt ausführen, obwohl noch kein Programm geladen ist oder es bereits zu Ende ausgeführt wurde.

MaxonSDB benötigt rct.library version 16!

Englisch: „MaxonSDB needs rct.library version 16!"
Um Meldungen und Dialogfenster darzustellen, ist die `rct.library` Version 16 oder höher unbedingt nötig. MaxonSDB kann sonst nicht gestartet werden.

MaxonSDB wirklich beenden?

Englisch: „MaxonSDB: quit really?"
Sie sind gerade dabei, das letzte offene Fenster von MaxonSDB zu schließen, oder haben den Befehl „Projekt/Debugger beenden" gegeben. MaxonSDB vergewissert sich, ob Sie ihn wirklich beenden wollen, da dies problematisch sein kann, wenn das zu debuggende Programm noch nicht beendet ist.

Overlays nicht erlaubt!

Englisch: „Overlays not supported!"
MaxonSDB kann keine Programme debuggen, die Overlays verwenden. Compilieren Sie Ihr Programm neu, so daß es ohne Overlays gebunden wird.

Programm endete normal. Rückgabewert: ...

Englisch: „Program terminated normally, returning ..."
Das Programm wurde zuende ausgeführt und kehrte zu MaxonSDB zurück. Entweder wurde die Funktion `exit()` aufgerufen oder die letzte Zeile der Funktion `main()` ausgeführt, oder auf Maschinensprachenebene mit dem ursprünglichen Wert in Register SP wurde ein `RTS`-Befehl ausgeführt.

Der dabei in Register D0 stehende Wert ist der Rückgabewert des Programms und wird im Dialogfenster angegeben. Nach der letzten Zeile von `main()` wird (sinngemäß) automatisch `exit(0)` aufgerufen, und das Argument beim Aufruf von `exit()` wird als Rückgabewert des Programms benutzt.

Sie können das Programm jetzt nicht mehr weiterlaufen lassen, aber die globalen Variablen, Funktionen und die Registerinhalte sind weiterhin vorhanden und können untersucht werden.

Programm läuft bereits!

Englisch: „Program already running!"
Sie wollten einen Einzelschritt ausführen, obwohl Ihr Programm bereits läuft.

Schreibfehler im Speicher!

Englisch: „Write-to-Memory Error!"
Beim Schreiben in den Hauptspeicher und anschließendem Prüflesen trat eine Abweichung auf. Sie können im ROM keine Haltepunkte setzen und keine Daten verändern, und Sie sollten mit MaxonSDB keine Daten in die Custom-Chip-Register schreiben.

Schreib oder Lesefehler!

Englisch: „Read/Write error!"
Beim Schreiben oder Lesen einer Datei trat ein Fehler auf. Die Ursachen können vielfältig sein, z.B. ein defekter oder voller Datenträger, eine unvollständige oder beschädigte Debugger-Datei oder ... Wenn Sie die Ursache beseitigt haben, dann versuchen Sie, den Befehl noch einmal auszuführen.

Speichermangel!

Englisch: „Out of memory!"
Ein Befehl konnte nicht ausgeführt werden, da zu wenig freier Hauptspeicher vorhanden ist. Abhilfe: Beenden Sie andere Applikationen, die Sie unvernünftigerweise gleichzeitig mit MaxonSDB laufen lassen (es ist immer ein Risiko, während einer Debuggersitzung noch etwas anderes mit dem Computer zu arbeiten), schließen Sie Bildschirme und/oder einige Fenster, löschen oder entfernen Sie die RAM-Disk, erweitern Sie den Hauptspeicher Ihres Rechners (1 MB ist Vorraussetzung!) und geben Sie dann den Befehl erneut.

Meldungen in der Statuszeile

Kein Quelltext verfügbar!

Englisch: „No Source available!"
Wenn Sie sich die Deklaration oder Definition einer Variable oder Funktion zeigen lassen wollen, zu der MaxonSDB keine oder nicht genügend Information in der `#?.mdbi`-Datei finden kann, erscheint diese Meldung.

8. Anhang

The Enforcer

Zumindest wenn Sie eingetragener Entwickler für den Amiga sind, kennen Sie den „Enforcer", der mittlerweile in mehreren Versionen existiert.

Mit diesem Werkzeug, das eine unbenutzte MMU vorraussetzt, können Sie in Ihren Programmen unzulässige Hauptspeicherzugriffe finden, die früher oder später zum Absturz des Programms führen würden.

Falls der Enforcer einen solchen Zugriff entdeckt, schickt er ausführliche Diagnosedaten über die serielle oder parallele Schnittstelle oder in ein Fenster, so daß diese Daten auf einem angeschlossenen Terminal oder Drucker sichtbar werden und man den Ort und Verursacher des Zugriffs feststellen kann.

MaxonSDB will uneingeschränkt die gleichzeitige Verwendung mit Enforcer erlauben und unterstützen. Da neuere Enforcer (ab Version 37) jetzt jeden unzulässigen Speicherzugriff melden, (ältere Versionen unterdrückten Meldungen über unzulässige Zugriffe mancher Programme, z.B. die des Debuggers „wack", so daß MaxonSDB sich bisher einfach in „wack" umbenennen konnte, um keine Meldungen zu erzeugen) muß sich MaxonSDB jetzt darum kümmern, daß er wirklich keine unzulässigen Hauptspeicherzugriffe macht.

Ob ein Speicherzugriff zulässig ist oder nicht, kann mit dem MMU-Befehl PTEST geprüft werden, so daß unzulässige Schreib- und Lesezugriffe unterlassen werden können. (Enforcer setzt ja ebenfalls eine MMU vorraus.) Dieser Test wird nur bei Speicherzugriffen der Benutzeroberfläche des MaxonSDB durchgeführt, die diese durchführt, um die von ihr darzustellenden Daten zu lesen, auf Benutzerwunsch zu ändern oder um Haltepunkte zu setzen. Dadurch wird jeder unzulässige Zugriff des zu debuggenden Programms weiterhin protokolliert, so daß der Zweck des Enforcers erhalten bleibt.

Dies ist nicht ganz ohne Probleme: Bei unzulässigen Lesezugriffen muß ein Ersatzresultat geliefert werden, damit die Benutzeroberfläche von MaxonSDB etwas zum Darstellen hat. Auch kann es bei der Verwendung von virtuellem Speicher vorkommen, daß auf einen bestimmten Speicherbereich (laut PTEST) kein Zugriff erlaubt ist, wenn MaxonSDB jedoch darauf zugreifen würde, würde er die korrekten Daten erhalten (sie werden dann automatisch von der Platte geholt). Da sich derzeit die Verwendung von virtuellem Speicher und Enforcer gegenseitig ausschließen, gibt es hierfür eine Einstellungen-Option.

Die Zusammenarbeit mit dem Enforcer geht aber noch weiter! Falls Enforcer Ihnen einen unzulässigen Zugriff durch Ihr Programm meldet, so setzen Sie einfach im Hauptfenster (mit „Neue Adresse...“) die dort gezeigte Adresse auf den Wert des von Enforcer protokollierten Programmzählers. Wenn Quelltext dazu existiert, zeigt MaxonSDB Ihnen die Stelle dann sogar im Quelltext, ansonsten können Sie z.B. einen Haltepunkt darauf setzen, und Ihr Programm hält das nächste Mal an, wenn es hier vorbeikommt, so daß Sie dann die dynamische Vorgeschichte des Problems herausfinden können.

Noch eine Warnung: Kickstart 1.2 und 1.3 und die dazugehörigen Programme machen häufig unzulässige Lese- und sogar Schreibzugriffe, so daß Sie hier durchaus mit „Enforcer-Hits“ rechnen müssen, wenn Ihr Programm Unterprogramme des ROMs aufruft, auch wenn Ihr Programm selbst völlig korrekt ist. Einige dieser fehlerhaften Zugriffe werden von Enforcer ignoriert und nicht protokolliert, aber andere, wie z.B. der in FindTask(„Name“) im Zusammenhang mit FFS-Partitionen (deren Name NULL ist), können seitenlange Protokolle erzeugen.

inline-Funktionen

Um die beim Aufruf und bei Rückkehr einer Funktion notwendigen Maschinenbefehle einsparen zu können, wurden in C++ die inline-Funktionen eingeführt. Für diese Funktionen wird nicht nur wie bei anderen Funktionen einmal Code erzeugt, der dann bei jedem Funktionsaufruf angesprungen wird. Statt dessen merkt sich der Compiler die Zeilen der Funktion und ersetzt (fast) jeden Aufruf der Funktion durch den Funktionskörper. (Es gibt spezielle Fälle, wo diese Ersetzung nicht möglich ist, z.B. beim Aufruf der Funktion über einen Funktionspointer oder bei virtuellen Funktionen.) Analog der `register`-Anweisung in normalem C ist die Beachtung des Schlüsselwortes `inline` keine Pflicht für den Compiler, er kann das Wort schlicht ignorieren.

Durch die Verwendung von `inline` bei Funktionen wird ein Programm im allgemeinen nicht kürzer, sondern höchstens schneller. Deshalb lohnt es sich auch nur dann, Funktionen mit der Eigenschaft `inline` zu versehen, wenn diese sehr kurz sind, z.B. weil sie nur einen Wert aus einem Klassenobjekt zurückliefern. Bei einem global optimierenden Compiler allerdings kann die Verwendung des Schlüsselwortes auch ein kürzeres Programm zur Folge haben, da sich diesem neue Möglichkeiten zur Optimierung über Funktionsaufrufe hinweg bieten.

Für einen Debugger hat die Verwendung von `inline` mehrere, unangenehme Konsequenzen, die daraus resultieren, daß aus einem kurzen Stück Quelltext viele verstreute Stellen im Code werden, und somit keine eindeutige Zuordnung „Quelltext zu Code“ mehr möglich ist.

Dadurch ist es nicht möglich, im Quelltext einen Haltepunkt in eine inline-Funktion zu setzen, und in Assembler müßten Sie einen Extra-Haltepunkt in jedes expandierte Stück setzen.

Eine inline-Funktion baut keinen Stack-Rahmen auf und verwendet die Register jedesmal anders, so daß ihre lokalen Variablen dem Debugger nicht zugänglich sind.

Der tiefere Sinn von inline-Funktionen, daß sie einem global optimierenden Compiler neue Optimierungsmöglichkeiten eröffnen, kann auch nicht genutzt werden, da ein solcher Compiler bei seiner Arbeit Quellzeilen und Variablen vertauscht, wegläßt und in ihrer Bedeutung verändert, und einen Quelltext-Debugger dabei vor kaum lösbare Probleme stellt. (Es ist sinnvoller, einen C++-Interpreter zu schreiben, als einen Debugger, mit dem man den Code im Quelltext nachvollziehen kann, den ein guter global optimierender C++-Compiler erzeugt).

MaxonSDB löst das Problem der inline-Funktionen folgendermaßen: Sie werden genau wie Makros behandelt. Dies bedeutet, daß der Aufruf einer inline-Funktion am Stück ausgeführt wird, auch wenn Sie „Schritt/Folge hinein" gewählt haben. Nur wenn Sie diese Stelle im Assemblermodus abarbeiten lassen, können Sie genauer untersuchen, was hier passiert.

Technische Einzelheiten

Die zwei Prozesse des MaxonSDB

MaxonSDB verwendet - anders als die meisten Debugger auf dem Amiga und auch auf sonstigen Rechnern - zwei Prozesse, die sich die Arbeit teilen. Ein Prozeß verwaltet komplett die Benutzeroberfläche und der andere arbeitet das zu debuggende Programm ab. Diese Zweiteilung bringt für Sie als Benutzer viele Vorteile:

Erstens ist es damit möglich, daß Ihr Programm seine Prozeß-Variablen völlig verändert (innerhalb der von AmigaDOS vorgesehenen Grenzen, natürlich) und z.B. das aktuelle Verzeichnis wechselt usw., ohne daß dies Auswirkungen auf die Benutzeroberfläche hat. Andere Source-Level-Debugger können in diesem Fall z.B. die Quelltexte nicht mehr finden!

Zweitens können Sie einen Programm, das in einer Endlosschleife oder `Wait(0L)` steckt, trotzdem unterbrechen und die Ursache untersuchen und evtl. beheben, und dann

das Programm ab der Unterbrechungsstelle (oder, mit Einschränkungen, jeder anderen Stelle) weiter ausführen, als wäre es nicht unterbrochen worden.

Drittens können Sie, während Ihr Programm z.B. zeitintensive Berechnungen durchführt oder wartet, zusätzliche Haltepunkte setzen oder überflüssige entfernen, globale Variablen betrachten oder ändern, und viele andere Dinge mehr, die bei anderen Debuggern nicht möglich sind.

Forbid() und Disable()

Falls ein durch diese Funktionen geschützter Bereich in Einzelschritten abgearbeitet wird, wird dieser Schutz bei jedem Schritt durchbrochen, da Ihr Programm innerhalb von MaxonSDB `Wait()` aufruft (aufrufen muß!), um den Befehl zum nächsten Einzelschritt von der Benutzeroberfläche von MaxonSDB zu erhalten. Dieses Problem würde aber genauso auftreten, wenn MaxonSDB nur einen Prozeß für seine Tätigkeit verwenden würde, da der Debugger ja letztendlich auf Intuition warten muß, um das von Ihnen eingegebene Einzelschrittkommando zu erhalten.

Ein mit obigen Funktionen geklammerter Programmabschnitt, der mit „Gehe" oder „Gehe bis Rückkehr" abgearbeitet wird, bleibt aber geschützt, falls Ihr Programm in diesem geschützten Abschnitt nicht auf einen Haltepunkt stößt. Auch innerhalb von Funktionen, die mit „Schritt über" am Stück ausgeführt werden, bleibt der Schutz durch `Forbid()` oder `Disable()` erhalten.

Die Prozessoren 68020,68030 und 68040

MaxonSDB berücksichtigt automatisch den in Ihrem Amiga gerade benutzten Prozessor (so wie es jedes Programm tun sollte und leicht kann). Darüberhinaus kümmert er sich (auch unter Kickstart 1.3!) um den Cache dieser Prozessoren, wenn er z.B. Haltepunkte im Speicher installiert oder entfernt. Unter Kickstart 2.0 verwendet er die hierfür vorgesehenen Funktionen der Exec-Library, so daß es auch auf Amigas mit zusätzlichem Cache keine Probleme geben sollte (falls es solche Amigas überhaupt gibt).

Allerdings können weder Maxon noch der Autor eine Garantie dafür übernehmen, daß MaxonSDB auf dem 68040 ganz ohne Probleme läuft, auch wenn wir MaxonSDB natürlich auch auf diesem Prozessor ausführlich getestet haben. Diese Einschränkung ist leider deshalb nötig, da sich die Behandlung von Prozessor-Exceptions auf dem 68040 derart umfangreich geändert hat, und Kickstart 2.x bisher keine Anstrengungen unternimmt, sich ausreichend darum zu kümmern.

Die Coprozessoren 68881 und 68882

MaxonSDB unterstützt die Verwendung von Amigas mit Coprozessoren, auch wenn er die Register und Befehle dieser Rechenknechte derzeit nicht darstellen kann. Der Debugger selbst verwendet die Coprozessoren nicht, und auch keine der diversen `mathXXXX.librarys`, er enthält die nötigen Umformungs- und Rechenfunktionen zum Umgang mit Fließkommazahlen aller Formate im Programm selbst.

Debuggen in Amiga-Libraries

Es ist (mit geringen Einschränkungen) möglich, Aufrufe von Funktionen der dem Amiga beiliegenden Libraries, die das eigene Programm macht, zu debuggen.

Die erste Einschränkung ist natürlich, daß Sie keinen Quelltext dazu haben, und daß Sie die Libraries nicht verändern können.

Die zweite, wichtigere Einschränkung ist die, daß MaxonSDB selbst auf die korrekte Funktion des Betriebsystems angewiesen ist. Deshalb dürfen Sie an Stellen, die auch von anderen Prozessen durchlaufen werden können, keine Haltepunkte setzen. Dies betrifft insbesondere die System-Libraries. Falls Sie dies doch versuchen, erscheint früher oder später eines (oder mehrere) der berühmten „Software Error - Task held"-Dialogfenster, und mindestens einer der System-Tasks - oder MaxonSDB selbst - stürzt ab. Zum Glück liegen die wichtigsten Libraries im ROM, wo Sie sowieso keine Haltepunkte setzen können.

Falls Sie eine Funktion mit Einzelschritten abarbeiten, so setzt MaxonSDB dabei keine Haltepunkte, und deshalb ist dies ohne Probleme möglich. Allerdings können Probleme auftreten, wenn Sie einen Bereich, der durch `Forbid()` oder `Disable()` geschützt ist, mit Einzelschritten abarbeiten, da dann der durch diese Funktionen aufgebaute Schutz unterlaufen wird.

MaxonSDB selbst verwendet mindestens die folgenden Libraries: `exec.library`, `dos.library`, `graphics.library`, `intuition.library`, `rct.library`, `hothelp.library`, `console.device`. Allerdings ist z.B. Intuition ohne Layers nicht lauffähig, und diese Liste kann sich täglich ändern.

Debuggen von eigenen Libraries

Falls Sie sicherstellen können, daß nur Ihr gerade debuggtes Programm Funktionen Ihrer selbsterstellten Library benutzt, können Sie durchaus Haltepunkte in diese Library setzen.

Das Abfangen von Prozessor-Exceptions

Bei der Entwicklung von MaxonSDB wurde darauf geachtet, daß es für ein zu debuggendes Programm möglichst keinen Unterschied machen sollte, ob es mit oder ohne Debugger läuft. Deshalb läuft ein von MaxonSDB gestartetes Programm in genau der Umgebung, in der MaxonSDB selbst gestartet wurde. Der einzige Unterschied ist der, daß `tc_TrapCode` (befindet sich in `struct Task`) auf eine andere Stelle zeigt, als sonst üblich. (Andere Debugger machen das genauso, es ist auf dem Amiga der einzig erlaubte Weg.) `tc_TrapCode` dient dazu, die auftretenden Prozessor-Exceptions dem gerade aktiven Task zuordnen zu können, so daß dieser Task selbst geeignet auf jede von ihm ausgelöste Prozessor-Exception reagieren kann. Prozessor-Exceptions treten z.B. bei Division durch Null oder Adress- oder Busfehlern, aber auch im Einzelschrittmodus oder bei der „Ausführung" eines Haltepunkts auf. Normalerweise zeigt dieser Pointer auf eine Funktion, die das „Software Error - Task held"-Dialogfenster zeigt und nicht mehr zurückkehrt. Falls Sie oder Ihr Compiler solche Fehler abfangen wollen, so muß Ihr Programm diesen Pointer mit der Adresse einer eigenen Maschinenspracheroutine überschreiben, damit das Programm davon etwas erfährt. (Falls Sie dies tun wollen, lesen Sie **unbedingt** ein gutes Buch über die Exceptions der MC-680x0-Familie und den Abschnitt über Exec im Amiga-ROM Kernel Manual: Libraries, und vergessen Sie insbesondere nicht, diesen Pointer am Schluß wieder auf den unsprünglichen Wert zu setzen!)

Durch die Veränderung dieses Pointers wird natürlich, falls Ihr Programm von einem Debugger ausgeführt wird, der vom Debugger in diesem Pointer eingetragene Wert überschrieben, so daß Ihre Routine statt die des Debuggers bei jedem Einzelschritt oder Haltepunkt aufgerufen wird. Um einem Debugger die Möglichkeit zu geben, Ihr Programm trotzdem zu kontrollieren, muß Ihre Routine mindestens bei den Prozessor-Exceptions „Illegal Instruction" (Haltepunkt, Nummer 4) und „Trace" (Einzelschritt, Nummer 9) die Kontrolle an die Routine übergeben, auf die `tc_TrapCode` vorher zeigte, und darf dabei den Stack nicht verändern.

SafeTrace

Falls der Prozessor, wenn Sie Ihr Programm in Einzelschritten ablaufen lassen, eine Division durch Null durchführen soll, so wird Exec Ihnen einen Guru $80000009 (Trace) präsentieren.

Dies hat komplizierte Ursachen: Erstens können manche Maschinenbefehle, wie z.B. der DIV-Befehl, wenn der Prozessor im Einzelschrittmodus (Trace-Mode) ist, zwei Prozessor-Exceptions gleichzeitig auslösen. Zweitens läßt Exec im Supervisor-Modus des Prozessors aus guten Gründen keine zusätzlichen Prozessor-Exceptions zu, sondern zeigt sofort einen Guru.

Da der Prozessor zuerst die Exception für Division durch Null bearbeitet und sich dann bereits im Supervisor-Modus befindet, meckert Exec (irreführenderweise!) die danach vom Prozessor bearbeitete Trace-Exception (Einzelschritt) an, und nicht Guru $80000005 (Division by Zero).

Dieser irreführende Guru tritt nicht nur bei Division durch Null auf. Jeder Maschinenbefehl, der eine Prozessor-Exception auslösen kann, ist hierfür verdächtig. Dies betrifft nicht nur die privilegierten Befehle, die von ordentlichen Compilern sowieso nicht verwendet werden, sondern auch DIV, CHK und TRAPV, die durchaus in von Compilern erzeugten Programmen vorkommen (können).

MaxonC^{++} z.B. erlaubt es Ihnen, Divisionen statt mit den (auch aus anderen Gründen) sichereren Bibliotheksroutinen mit den schnelleren DIV-Befehlen durchzuführen. Andere Compiler (z.B. für Pascal oder Modula-2) verwenden CHK und TRAPV-Maschinenbefehle, um Index- oder Rechenfehler effizient abfangen zu können. Falls Ihr Programm fehlerhaft ist und an diesen Stellen mit Einzelschritten abgearbeitet wird, wird dieser Guru auftreten.

In solch einem Fall werden Sie wohl die Schuld Ihrem Debugger zuschreiben, da Exec ja immer Guru $80000009 (Trace) zeigt, aber es gibt für Ihren Debugger keine von Commodore zugelassene Methode, dies zu vermeiden. Aber: Ein Programm, das unter einem Debugger zu diesem Guru führt, ist fehlerhaft, und würde ohne Debugger warscheinlich zum „Software Error - Task held"-Dialogfenster führen. („Wahrscheinlich" deshalb, weil das Programm ja prinzipiell die Möglichkeit hat, die Prozessor-Exceptions abzufangen. Siehe: „Das Abfangen von Prozessor-Exceoptions")

Nun, es gibt eine (von Commodore NICHT unterstützte!) Methode, diesen Guru zu vermeiden: (Vorsicht, jetzt wird es noch technischer, aber Sie brauchen diesen Abschnitt nicht unbedingt zu verstehen, um das Programm SafeTrace erfolgreich anwenden zu können!) Ein Programm könnte sich direkt in den Exception-Vektor für Trace eintragen, und im Falle einer Trace-Exception, die im Supervisor-Modus auftritt, diese Exception einfach ignorieren (`RTE`), wodurch der Handler der anderen Exception zum Zug kommt.

Wenn dieser Handler den Supervisor-Modus (durch RTE) wieder verläßt, wird ein zu Beginn bestehender Trace-Modus automatisch wieder eingeschaltet.

Falls es sich um eine ganz normale Trace-Exception im User-Modus handelt, würde das Programm genau dasselbe tun, was Exec in allen bisher bekannten Versionen hier auch macht, nämlich den tc_TrapCode des aktuellen Tasks aufrufen, nachdem es 9L auf den Stack gelegt hat.

Das Problem dabei ist, daß Commodore es nicht unterstützt, wenn ein Programm die Tabelle der Exception-Vektoren ändert. Deshalb kann ein solches Programm in der Zukunft Probleme verursachen.

Damit MaxonSDB „zukunftskompatibel" bleibt, haben wir den oben beschriebenen Algorithmus **nicht** in MaxonSDB eingebaut. Da es allerdings trotzdem das Ziel von MaxonSDB ist, auch Fehler aus solchen Programmen entfernen zu können, die die DIV-Befehle des Prozessors verwenden, haben wir den Algorithmus in ein winziges Extra-Programm verpackt, das vor MaxonSDB aufgerufen werden kann. Dieses Programm können Sie dann leicht weglassen, wenn Sie eine Exec-Version benutzen, bei der diese Methode vielleicht mehr schadet als nützt (Vielleicht Kickstart 5.0 mit virtuellem Speicher? Wer weiß, was Commodore in Bezug auf die Exceptions noch alles vorhat...).

Dieses Programm nennen wir „SafeTrace". Es installiert einen neuen Trace-Handler, der genau das oben Beschriebene macht. Dabei kümmert sich SafeTrace natürlich um eventuell vorhandene Prozessor-Caches und ein eventuell verwendetes VBR-Register. Auch eine Versuch, SafeTrace mehrfach zu installieren, wird abgefangen.

Es ist nicht möglich, den von SafeTrace installierten Handler ohne Bootvorgang wieder zu entfernen, dies sollte aber kein Problem darstellen.

9. Das Format der Debugger-Dateien

Hier wird in aller Kürze erklärt, was in den geheimnisvollen „#?.mdbg" und „#?.mdbi"-Dateien enthalten ist, und wie es kodiert ist.

Zu jeder Übersetzungseinheit (im Folgenden **„Modul"** genannt) erzeugt der Compiler auf Wunsch eine „#?.mdbi"-Datei. (*cppc* kann man mit den Optionen -b oder -bi dazu bringen, und in der integrierten Umgebung MCPP gibt es dafür den Menüpunkt „Optionen/Debugger-Datei".) Der eingebaute Linker erzeugt dann automatisch noch eine „#?.mdbg"-Datei. Bei anderen Compilern schauen Sie bitte in deren Handbuch nach.

Wofür diese vielen Dateien? Nun, man könnte auch alles in eine Datei packen, aber dann müßte der Compiler entweder alles auf einmal übersetzen, damit er den Überblick behält (das wird auf Dauer bei größeren Projekten zu lahm) oder Zwischendateien anlegen, in denen jede Information noch ein zweites Mal steht, und diese dann beim Linken einlesen, zusammenfassen und in eine Extra-Datei schreiben. Da quillt dann schnell die Platte über, und das Linken, das bei großen Projekten eh der zeitaufwendigste Teil ist, wird noch langsamer.

Also haben wir uns für diese Aufteilung entschieden.

Der Inhalt der Dateien

Ein Source-Level-Debugger braucht recht umfangreiche Informationen über den Quelltext, viel mehr als ein sogenannter „symbolischer Debugger", den nur die Startadressen von Funktionen und Variablen, und nicht mal deren Größe, geschweige denn deren Typ interessiert. Zu diesen Informationen gehören:

Bezeichner
Der Debugger soll ja die Dinge beim Namen nennen und erkennen können, deshalb muß der Name jeden Objekts in der Datei abgelegt werden.

Objekte
Es gibt vielerlei Objekte, die der Compiler einzeln unterscheidet, und auch der Debugger sollte diese kennen. Objekte sind zum Bleistift:

Funktionen, Prozeduren, globale und lokale Variablen, Funktionsargumente, Namen für Typen (typedef-Namen und Typ-Tags (das „Window" in struct Window)), Klassen-Elemente und Elemente einer Struktur.

Typen
Der Typ eines jeden Objekts muß erkennbar sein, und viele Objekte haben den gleichen Typ.

Enum-Werte
Es ist sehr sinnvoll, wenn eine *enum*-Variable im Klartext angezeigt werden kann, denn schließlich hat man als Programmierer absichtlich einen Namen und keine Zahl im Quelltext verwendet.

Scope-Bereiche
Um die Sichtbarkeitsregeln des Compilers bei der Auswertung von Ausdrücken nachbilden zu können, muß Information über die Schachtelung und Beginn und Ende jedes Scopes vorhanden sein.

Quelldateien
Der Debuggger muß jede verwendete Quelldatei kennen, und wissen, wo welches Objekt deklariert oder definiert wurde.

Statements
Da der Debugger scheinbar „Quellzeilen" ausführen soll (und nicht Maschinenbefehle), muß er wissen, zu welchem Statement welcher Code gehört, und umgekehrt. Dabei ist auf dem Amiga auch wichtig, in welchem Hunk der ausführbaren Datei der Code liegt.

Das Format der Dateien

Als Format der Debugger-Dateien wurde ein neues, kompaktes IFF entworfen, das „*FORM MDBI*", wobei MDBI als Abkürzung für „Maxon DeBugger Information" stehen soll. Dieses Format wird für die `#?.mdbi`-Dateien verwendet und wird unten ausführlich erklärt. Die `#?.mdbg`-Dateien sind *ASCII*-Klartext, da sie recht klein sind und von Hand durchaus gepatcht werden können sollen. Bei allen Dateiformaten wurde darauf geachtet, daß sie auch für andere Sprachen wie z.B. *Pascal* oder *Modula-2* gut geeignet sind, da Maxon hier noch einiges vor hat...

Fangen wir mit dem Übersichtlicheren an:

Das Format der *#?.mdbg*-Datei

Diese Datei soll den Debugger über die Aktivität des Linkers aufklären. Deshalb enthält sie Information über die eingebundenen `#?.o`-Dateien, und was aus ihnen geworden ist. Darüber hinaus wird dem Debugger mitgeteilt, wo sich die Hauptprozedur des Programms befindet. In C ist dies zwar immer `main()`, aber z.B. MaxonC^{++} hat noch

zusätzlich `wbmain()`, und in Pascal hat diese Prozedur gar keinen (festen) Namen, je nach Compiler.

Das Format dieser Datei ist zeilenorientiert. Dabei entscheidet das erste Zeichen einer Zeile über ihre Bedeutung.

Zeilen, die mit Semikola ("Herr Ober, ein halbes Cola, bitte") oder Sternen (; oder *) beginnen, werden komplett ignoriert.

Für jedes Modul, für die eine `#?.mdbi`-Datei erzeugt wurde, werden ein paar (wenige) aufeinanderfolgende Zeilen abgelegt.

Diese Block von Zeilen sieht so aus: Die erste Zeile enthält den Namen der Objektdatei in Hochkommas (") eingeschlossen, wobei ein relativer Pfad zum Ort der #?.mdbg-Datei angegeben werden sollte, falls sie nicht sowieso im gleichen Verzeichnis liegt. Den Namen der #?.mdbi-Datei erhält man, indem man das „.o" (oder wie die Endung auch immer sei) am Ende durch „.mdbi" ersetzt.

Darauf folgt für jeden Hunk (z.B. `"CODE", "DATA", "BSS"`) der Objektdatei eine Zeile, in der vermerkt wird, in welchen Hunk der ausführbaren Datei der Hunk eingefügt wurde und mit welchem Offset. Diese Zeile beginnt mit einer Zahl (um exakt zu sein: mit einer Ziffer), der neuen Hunk-Nummer, dann folgt ohne Leerzeichen ein Komma, und dann nach eventuellen Leerzeichen noch eine Zahl, das Offset des Hunks im neuen Hunk.

Anschließend beginnt der Zeilenblock des nächsten Moduls...

Abschließend kommen noch eine oder mehrere Zeilen, die mit einem Ausrufezeichen ! beginnen. In diesen Zeilen werden die Startadressen der *wbmain*-Funktionen abgelegt, und zwar als eventuell Leerzeichen, Hunknummer in der ausführbaren Datei, Komma ohne Leerzeichen davor, eventuell Leerzeichen, Offset im Hunk.

Tip: Da der MaxonSDB alle in einer `#?.mdbg`-Datei erwähnten `#?.mdbi`-Dateien in den Hauptspeicher lädt, kann man bei Rechnern mit geringem Speicherausbau und großen Projekten manche Zeilenblöcke auskommentieren, so daß diese zwar nicht debuggt werden können, aber auch kein wertvolles RAM belegen. Es ist aber nicht sinnvoll möglich, diese `#?.mdbi`-Dateien bei späterem Bedarf nachzuladen, da dann die Hunk/Offset-Information fehlt, und der Debugger die Daten nicht zuordnen kann.

Das Format der *#?.mdbi*-Datei

Diese Datei enthält alle Informationen zu einem Modul. Da dies eine ganze Menge sein kann, ist diese Datei binär und kann **nicht** im Klartext gelesen werden.

Es handelt sich, wie oben erwähnt, um eine IFF-Datei mit dem *FORM*-Typ *MDBG*. *CAT* und *LIST* werden nicht benötigt, denn es gibt keine globalen „Properties", und diese beiden Chunks sind deshalb auch nicht erlaubt. Innerhalb des FORMS befinden sich, wie üblich, **Chunks**. Da keine der von Commodore-Amiga veröffentlichten Chunks für unsere Zwecke passend war, haben wir neue Chunk-Typen entwickelt, die sich im Aufbau alle ähnlich sind, aber grundverschiedene Bedeutung haben.

Diese Chunks tragen die Namen **STRG**,**TYPE**, **ILST**, **ENUM**, **SCOP**, **HUNK** und **SRCS**.

Da die Daten für den Debugger eine hochgradig verzeigerte Grundstruktur haben, und man Zeiger nicht einfach in eine Datei schreiben kann, haben wir diese Struktur auf mehrere „Array of Record"s abgebildet, wobei dann Indices in diese Arrays die Pointer ersetzen und der Index 0 den NULL-Pointer. Die Indices sind alle als UWORD deklariert, 65535 verschiedene Records von jedem Chunk-Typ in einer Datei sollten in allen Fällen ausreichen.

Damit die Records nicht einfürallemal festliegen müssen, und damit nicht unnötige Daten geschrieben werden müssen, wird bei (d.h. vor) jedem Array noch der Startindex (niemals 0!) und die Elementgröße (jeweils ein UWORD) abgelegt (der letzte gültige Index in das Array läßt sich dann aus der Chunkgröße und diesen beiden Werten berechnen). Die Elementgröße muß im Hinblick auf den 68000er Prozessor immer gerade sein, damit kein Element auf ungeraden Adressen beginnt.

Aufgrund dieser beiden Angaben ist es möglich und sinnvoll, daß ein Chunk mehrfach, z.B. mit unterschiedlicher Elementgröße, auftritt, solange die Indices der Arrays sich nicht überschneiden, solange die Indices also eindeutig sind. Es dürfen dabei Lücken zwischen den Arrays auftreten, z.B. zwei Arrays STRG, eines mit Indices von 15 bis 367 und noch eines von 4096 bis 6773, sofern die Indices dazwischen (und außerhalb sowieso) nicht angesprochen werden!

Im allgemeinen (es gibt auch Ausnahmen) hat ein Chunk also folgenden symbolischen Aufbau:

```
struct MDBIChunk
{ ULONG  NAME;
  ULONG  Größe;
  UWORD  Index_des_Ersten_Records;
  UWORD  Recordgröße;
  char[] Daten;
};
```

Nun zu den Chunks im einzelnen:

Chunk STRG

Dieser Chunk enthält Strings. Darunter fallen: Bezeichner, Pfad und Dateinamen und was sonst noch so an Strings anfällt. Da die Größe der Strings sehr unterschiedlich sein kann, ist dieser Chunk gleich die erste Ausnahme vom oben beschriebenen regelmäßigen Aufbau: Das erste UWORD des Chunks (das nach der Chunk-Größe) gibt wieder den Index des ersten Strings an (d.h. unter diesem Index wird von außen auf den String verwiesen), aber statt UWORD Recordgröße steht UWORD Stringanzahl. Die danach folgenden Daten sind dichtgepackte Strings im Format ASCII-Zero, d.h. jeder String hat noch ein NULL-Byte angehängt und darf innendrin keines enthalten. Wenn man den erste String des Arrays 1. String nennt, läßt sich der 5. String dieses Arrays finden, indem man vom Anfang der Strings an die NULL-Bytes zählt, der String beginnt dann nach dem 4. NULL-Byte und endet vor dem 5. (MaxonSDB macht diese aufwendige Zeichenzählerei natürlich nur 1x und legt dabei die Startadressen der Strings in einem Extra-Array ab, wobei die Angabe der Anzahl der Strings hierfür recht nützlich ist...).

Die ersten 16 String-Indices haben eine spezielle, feste Bedeutung und dürfen deshalb nicht in einem Chunk STRG definiert werden: Index 0 ist, wie immer, der NULL-Pointer, d.h. die Angabe für „Kein String bekannt". Index 1 bis 4 erhalten intern im Debugger folgende Zuordnung:

1 eine Basisklasse („<BASE>")
2 die „Virtuelle-Methoden-Tabelle" („<VMT>")
3 das aktuelle Klassenelement („this" in C++)
4 dito, aber für Pascal : „self"

Index 5 bis 15 sind reserviert und werden von Maxon für ähnliche Zwecke verwendet werden.

Die Indices 16 bis 255 sind schließlich noch erwähnenswert, da manche andere Records nur ein UBYTE für einen Stringverweis vorsehen, und diese Records dann natürlich nur diese Indices ansprechen können.

Noch ein Hinweis zu Destruktoren und Operator-Funktionen: Der Name des Destruktors der Klasse List heißt „~List" und wird auch so als String abgelegt, und z.B. der [ ]-Operator heißt „operator [ ]".

Chunk TYPE

Alle Typ-Informationen werden in Chunks „TYPE" abgelegt. Da der Umfang und Inhalt einer Typinformation sehr unterschiedlich sein kann. (Ein int wird durch eine Zahl ausreichend gekennzeichnet, und eine normale Klasse benötigt mindestens einen Tag-Namen, eine Member-Liste, usw., sind die Records dieses Arrays am besten als variante Records darstellbar (falls Strukturen und Unions auf einer ungeraden Adresse beginnen könnten, gilt folgender Aufbau, also nehmen wir dies der Klarheit halber mal an):

```
struct  MDBI_TYPE
{ UBYTE Tag;          /* Zur Bestimmung des varianten Teils, s.u. */
  union
  {
    struct
    { UBYTE Zero;     /* Immer NULL */
      UWORD Type;     /* Der Pointer zeigt auf diesen Typ */
    } pointer;
    struct
    { UBYTE Zero;     /* Immer NULL */
      UWORD Type;     /* Die Referenz zeigt auf diesen Typ */
    } reference;
    struct
    { UBYTE Zero;     /* Immer NULL */
      UWORD Type;     /* Der bptr zeigt auf diesen Typ */
    } bptr;
    struct
    { UBYTE What;     /* Bit 0: const, Bit 1: volatile */

      UWORD Type;     /* Ansonsten dieser Typ */
    } volatile_const;
    struct
    { UBYTE Type;     /* GrundTyp (BYTE bis ULONGLONG) */
      UWORD Name;     /* Name des enum-Typs */
    } enum;
    struct
    { UBYTE Zero;     /* Immer NULL */
      UWORD TabellenAnzahl; /* Anzahl der Einträge in der VMT
    } vmt_pointer;
    struct
    { UBYTE Zero;     /* Immer NULL */
      UWORD IndexType; /* Typ des Index (MUSS Subrange oder
                          BYTE..ULONGLONG oder unbe-
                          kannt(=0) sein!) */
      UWORD ElemType;  /* Typ eines Arrayelements */
```

```
    ULONG ElemSize;   /* Größe eines Arrayelements (zum
                         Entlangwandern im Array
                         Hinweis: Die Startadresse eines Arrays
                         ist die Adresse des Ersten vorhandenen
                           Elements! */
    } array;
  struct
  { UBYTE Type;     /* GrundTyp (BYTE bis ULONGLONG) */
    LONG Low;       /* Unterer gültiger Wert */
    ULONG High;     /* Oberer gültiger Wert */
  } subrange;
  struct
  { UBYTE Zero;     /* Anordnung der Bits, ect... ?????? */
    UWORD IndexType; /* Set of was? (MUSS Subrange oder
                        BYTE..ULONGLONG sein! Nicht 0!) */
    LONG Low;       /* Nummer des ersten Bits, nötig wg.
                       Rundung bei KickPas. */
    UWORD Anz;      /* Anzahl der Bits in der Menge,
                       0 bedeutet 65536 */
  } set;
  struct
  { UBYTE Size;     /* string [size], normalerweise 255 */
                    /* Eventuell weitere Daten, die
                       noch definiert werden müssen! */
  } pascalstring;  /* Aber auch für BCPL geeignet */
  struct
  { UBYTE ResultPlace; /* Wo liegt das Funktions-
                          ergebnis? s.u. */
    UWORD ResultType;  /* Typ des Ergebnisses der Funktion */
    UWORD ArgList;     /* Liste der Funktionsargumente */
    UWORD ClassType;   /* 0, oder bei Memberfunktionen
                          der Klassen-Typ */
    UBYTE Stack;       /* Bit 0: Wer räumt die Argumente weg?
                          0: Caller, 1: Funktion selbst. */
                       /* Bit 1: Benutzt die Funktion
                          LINK/UNLK A5?
                          0: ja, 1: nein. Restliche Bits NULL */
    UBYTE Zero;        /* Immer NULL */
  } function;
  struct
  { UBYTE What;        /* Bit 1-0; 0=struct/record, 1=class/
                          object, 2=union; 3=undefiniert. andere
                          Bits NULL */
    UWORD Name;        /* Name der Klasse */
    ULONG Size;        /* sizeof(class Name) */
    UWORD MemberList;  /* Liste der Mitglieder */
  } classinfo;
} };
```

Der Tag, der angibt, um welches Record es sich handelt, kann folgende Werte annehmen:

0 unbekannter Typ / kein Typ
1 void (0 Byte) Void ist ein gültiger Typ!
2 UBYTE (1 Byte)
3 BYTE (1 Byte)
4 UWORD (2 Byte)
5 WORD (2 Byte)
6 ULONG (4 Byte)
7 LONG (4 Byte)
8 unsigned long long (8 Byte)
9 signed long long (8 Byte)
10 FFP-Fließkomma (4 Byte)
11 IEEE-single (4 Byte)
12 IEEE-double (8 Byte)
13 6888x-extended (12 Byte)
16 char (1 Byte, z.B. in Pascal)
17 wchar (2-Byte-Character)
18 lchar (4-Byte-Character)
19 bool (1 Byte, Wert 0 ist False, alles andere True)
20 BOOL (2 Byte, Wert 0 ist False, alles andere True)
21 bool (4 Byte, Wert 0 ist False, alles andere True)
31 „..." in C/C++

32 pointer (4 Byte), s.o.
33 reference (4 Byte), s.o.
34 bptr (4 Byte), (Pointer durch 4 geteilt) s.o.
40 enum, s.o.
41 array, s.o.
42 subrange, s.o.
43 set, s.o.
44 Pascal-String, s.o.
45 Funktionstyp, s.o.
50 class/struct/union ect, s.o.
51 vmt_pointer, (4 Byte), Zeiger auf die VMT, s.o.
70 volatile_const, s.o.

Alle anderen Werte sind reserviert und können jederzeit von Maxon Computer definiert werden!

Der Ort, wo die Funktion ihr Ergebnis ablegt, wird wie folgt angegeben:

0 void, kein Ergebnis
1 in Register d0
2 in Register a0
3 in Register fp0
4 an $8(A5) liegt ein Pointer, der auf das Ergebnis zeigt.
254 Konstruktor, hat kein Ergebnis.
255 Destruktor, hat kein Ergebnis.

Wie immer sind alle anderen Werte reserviert und können urplötzlich in neuen oder alten Maxon-Produkten auftreten!

Zur Vereinfachung sowohl für den Compiler, als auch für den Debugger, werden die Typ-Indices 0 bis 127 reserviert, und zum Teil wie folgt definiert:

Index 0 bis 31:	direkt analog dem Typ-Tag, d.h. Index 4 ist Typ „UWORD", usw.
Index 32 bis 63:	Pointer auf Typ 0..31, d.h. Index 41 ist „long long*"
Index 64 bis 127	bleibt erstmal reserviert.

Chunk ENUM

Um die Werte einer enum-Variablen im Klartext darstellen zu können, muß MaxonSDB die Namen der Werte kennen. Diese Informationen werden in diesem Chunk abgelegt.

Die Records des Arrays sind sehr einfach aufgebaut:

```
struct MDBI_ENUM
{ UWORD Name;  /* Name dieser Konstanten */
  UWORD Typ;   /* Gehört zu diesem Enum-Typ */
  LONG oder LONGLONG Wert; /* Was es ist, erkennt man an
                              der Recordgröße */
};
```

Records dieses Hunks haben normalerweise eine Größe von 8, aber falls tatsächlich enums mit LONGLONG-Werten benutzt werden eine Größe von 12 (und sind dann nicht ganz „Motorola-like" definiert).

Keine Indices sind reserviert, da keine Indices hierdrauf verwendet werden.

Chunk SCOP

Die Schachtelung von Gültigkeitsbereichen (z.B. in Pascal durch begin/end) ist auch für den Debugger von Interesse, wenn er Ausdrücke im aktuellen Gültigkeitsbereich auswerten oder die aktuellen Variablen zeigen soll.

```
struct MDBI_SCOP
{ UWORD  Parent;          /* Umschließender Gültigkeitsbereich */
  UWORD  Objekte;         /* Liste der hierdrin definierten Objekte */
  ULONG  QuellPosStart; /* Start des Scopes im Quelltext */
  ULONG  QuellPosEnde;  /* Ende des Scopes im Quelltext */
};
```

Das Format einer Quellposition wird unten erklärt. Index 1 ist der Gültigkeitsbereich, der die ganze Datei umfaßt. Dieser Scope hat (in normalen Computersprachen) als einziger keinen umschließenden Scope. Objekte, die nur importiert werden (`extern int errno;`) haben einen speziellen CodeOrt (s.u.).

Chunk HUNK

Hier werden nochmal alle Objekte der Adresse nach aufgelistet, die statisch in den Hunks abgelegt sind. Dieser Chunk ist eigentlich überflüssig, da grundsätzlich redundant, aber er kann bei manchen Suchvorgängen im Debugger hilfreich sein. Die Records sind hier nur einfache UWORDs, und statt Index des ersten und der Recordgröße wird hier die Hunknummer (UWORD) und der Name des Hunks (UWORD) (verweist auf `"CODE"`, `"DATA"`, `"BSS"`) abgelegt, dann folgen die UWORDs der Objekte.

Chunk SRCS

Damit MaxonSDB die Quelldateien anzeigen kann, werden hier alle vom Compiler eingelesenen Quelldateien aufgelistet. Da viele Dateien in immer den gleichen Verzeichnissen liegen, wurde eine Trennung zwischen Pfadname und Dateiname gemacht. Es ist aber auch möglich, PfadName auf 0 zu setzen und den Pfad im Dateinamen abzulegen, denn Pfad- und Dateiname werden im Debugger einfach aneinandergehängt, ohne ein „/ " oder ähnliches dazwischen einzufügen.

```
struct MDBI_SRCS
{ UWORD  PfadName;          /* Pfadname, meistens relativ zur
                               "#?.mdbi"-Datei  */
  UWORD  DateiName;         /* Dateiname bzgl. des Pfads */
  ULONG  LastChange;        /* EOR über die 3 Langworte des DateStamps
                               der Datei, oder Dateilänge in Bytes */
  UBYTE  SourceLanguage;    /* Quellsprache dieser Datei, s.u. */
  UBYTE  CompilerName;      /* Ganz normaler Namensindex (im
                               Bereich 15 bis 255) */
};
```

Zu LastChange: Damit soll der Debugger erkennen können, ob die Datei seit dem letzten Compilerlauf verändert wurde. Hierfür bietet sich die Abfrage der Uhrzeit an, wann die Datei zuletzt geändert wurde, da diese automatisch von AmigaDOS verwaltet wird. Das Programm *make* macht im Prinzip auch nichts anderes. Da aber auf „uhrlosen" Systemen wie einem Standard-Amiga 500 die aktuelle Zeit im allgemeinen nicht stimmt, kann der Compiler dort die Länge der Datei zum Übersetzungszeitpunkt (gemessen in Bytes) eintragen, auf Amigas mit Uhr erlaubt ein Datumsvergleich aber eine zuverlässigere Erkennung von Veränderungen. (Ein Test mit Hilfe von Checksummen oder ähnlichem würde einen Compiler deutlich verlangsamen. Dies steht in keinem Verhältnis zum erzielten Genauigkeitsgewinn, denn normalerweise ist Ihnen ja bekannt, welche Dateien Sie verändert haben.)

Für SourceLanguage sind folgende Werte bereits definiert und werden vom MaxonSDB bei der Auswertung von Ausdrücken im aktuellen Kontext verwendet:

0 C++
1 C
2 Kick-Pascal
3 Assembler (ja, auch ein Assemblerprogrammierer kann von MaxonSDB profitieren!)
4 Modula-2

Chunk ILST

Ständig braucht der Compiler, und auch der Debugger, eine Liste von Objekten. Nun, hier ist sie! Der Name „ILST" stammt ursprünglich von „Instance-LiST", aber „Instanz" klingt etwas zu sehr nach Gerichtsverfahren, darum verwende ich hier in dieser Dokumentation lieber den Ausdruck „Objekt", auch wenn das Wort mindestens fünf andere Bedeutungen zusätzlich hat. Dieser Chunk ist ein Beispiel, wo MaxonC++ zwei Chunks mit verschiedenen Recordgrößen ablegt, einen für Variablen (14 Byte) und einen für Funktionen (20 Byte).

```
struct MDBI_ILST
{ UWORD  NextInstance;  /* Objekte werden durch Verkettung
                           in Listen eingereiht */
  UWORD  Name;          /* Falls das Objekt einen Namen trägt */
  UWORD  Typ;           /* Typ des Objekts, notwendig */
  ULONG  CodeOrt;       /* Wo startet das Objekt im Speicher, s.u. */
  ULONG  QuellPosition; /* Wo wurde das Objekt definiert/
                           deklariert und, falls der Typ ein
                           Funktionstyp ist,(und somit dieses
                           Objekt eine Funktion!), gibt es noch
                           mehr: */
  UWORD  Scope;         /* Dies ist der Scope des Funktionskörpers,
                           und enthält die lokalen Variablen und die
                           Funktionsargumente */
  UWORD  PosnByteIndex; /* Offset in Chunk POSN, CodeOrt
                           + QuellPosition bestimmen die
                           Startwerte (s.u.) */
  UWORD  CodeSize;      /* Größe der Funktion in Bytes.
                           (Bei Variablen kann man die Größe am
                           Typ erkennen.) */
};
```

Chunk POSN

Um den aktuellen Quelltext darstellen und Quelltextzeilen am Stück ausführen zu können, benötigt der Debugger eine genaue Zuordnung zwischen Maschinencode und Quelltext. Dafür gibt es diesen Hunk, in dem die Daten als Byte-Folge abgelegt sind. Einen ersten Index oder eine Recordgröße gibt es deshalb hier nicht!

Es wird immer erst eine neue Codeposition und dann dazu eine neue Quellposition kodiert angegeben, solange, bis es über den Code oder Quelltext nichts mehr auszusagen gibt, und somit dieser Chunk zuende ist. Die Codeposition nimmt dabei monoton zu, aber die Quelltextposition kann beliebig (!) springen. Die Bytefolge kann prinzipiell vom Anfang bis zum gewünschten Ort gelesen werden, ein Quereinstieg ist aber möglich, wenn die auf diese Weise akkumulierten Code- und Quellpositionen anderswo beschafft werden können (z.B. bei Funktionsobjekten s. o.).

Zu Beginn steht der Codezähler auf Byte 0 des Hunks CODE (dieses Moduls), und die Quellposition ist undefiniert und wird deshalb gleich definiert.

Die aktuelle Codeposition wird wie folgt entschlüsselt:

Lies ein Byte.
Falls der Wert des Bytes 0 bis 250 ist, erhöhe die CodePosition um diesen Wert (ungerade Werte sind bei 680x0-Prozessoren nicht sinnvoll!).
Falls der Wert 255 ist, lies die zwei folgenden Bytes wie ein 68020, mache daraus ein UWORD und addiere dies auf die CodePosition.
Die Werte 251 bis 254 sind reserviert und dürfen nicht auftreten bzw. müssen angemeckert werden.

Die Quellposition wird analog dazu entschlüsselt, ist aber etwas komplizierter, da auch die Datei gewechselt werden kann:

Lies ein Byte.
Falls dies den Wert 254 hat, lies die 4 folgenden Bytes wie ein 68020, mache daraus die neue Quellposition und vergiß die alte.
Falls der Wert 255 ist, lies die zwei folgenden Bytes wie ein 68020, mache daraus ein WORD (signed!), und addiere dies auf das aktuelle Quellort-Offset.
Falls der Wert 0 bis 250 beträgt, addiere diesen Wert auf das Quellort-Offset.

Jetzt wird es Zeit, etwas über die Codierung von Quellpositionen zu sagen:

Die Codierung von Quellpositionen

Prinzipiell benötigt eine Quellposition nur einen Dateispezifizierer und ein Offset innerhalb der Datei.

Da es wünschenswert ist, daß z.B. ein `for(a=9;a;a--) b++;`-Statement nicht nur als Ganzes ausführbar, sondern jeder seiner 4 Bestandteile einzeln schrittweise abgearbeitet werden kann, auch wenn alle in einer Zeile stehen, sollte das Offset als Zeichen-Offset abgelegt werden. Andererseits ist es bei der Verwendung des #line 11 „datei.y"-Präprozessorkommandos sinnvoll, daß der Debugger die *yacc*-Quelldatei zeigt und nicht die von yacc erzeugte, eventuell kaum lesbare „C"-Datei. Bei dieser *yacc*-Quelldatei weiß der Compiler aber nur von Zeilennummern und kennt die Zeilenlängen nicht.

Da eine Quellposition nicht mehr Platz als ein LONG einnehmen sollte, und eine Quelldatei einerseits recht oft größer als 65535 Zeichen ist (dies ist ja nicht Turbo-Pascal!), und andererseits die Amiga-Includes bereits über 220 Stück zählen, und noch ein Bit zur Unterscheidung zwischen Zeilen- und Zeichenoffset notwendig ist, ergab sich folgendes krumme, aber trotzdem effiziente Format: (jeder Buchstabe steht für ein Bit)

ZDDDDDDD DDDDOOOO OOOOOOOO OOOOOOOO

Z (1 Bit)= 1, falls das Offset Zeilennummern angibt, sonst 0.
D (11 Bit) Index in SRCS, dies erlaubt über 2000 Dateien. 0 ist wie üblich: ungültige Datei.
O (20 Bit) Offset. Damit kann eine Datei ein Megabyte groß sein oder über eine Million Zeilen haben.

Die Codierung von Codepositionen

Eine Codeposition ist etwas einfacher aufgebaut und kann hier deshalb in Pseudo-Hexadezimal statt in Bits angegeben werden:

$TTOOOOOO

T (8 Bit) Um was für einen Ort handelt es sich? (s.u.)
O (24 Bit) Vorzeichenbehaftetes Offset bezüglich der mit $TT angegebenen Basis. Offset kann also (als ULONG betrachtet) folgende Werte annehmen: $FF800000 bis $FFFFFFFF und $0 bis $7FFFFF

$TT kann (als BYTE betrachtet) folgende Werte annehmen:

-1 bis
-128 Die Basis ist die Basisadresse des Hunks mit der Nummer (1-$TT), d.h. -1 steht für Hunk 0 des Moduls und -128 für Hunk 127 des Moduls, falls es diese Anzahl von Hunks jemals in einer Übersetzungseinheit geben sollte. (Der ~-Operator (Tilde) leistet bei der Umrechnung gute Dienste.)
0 Die Basis ist nicht bekannt und somit dieser Codeort nicht gültig.
1 Das Objekt liegt absolut im Speicher. Offset ist auch hier leider vorzeichenbehaftet, so daß z.B. die Custom-Chips hiermit nicht erfaßt werden können. (Es gilt ohnehin: Wenn jemand ein Inspect-Fenster (z.B. ein WORD-Array) oder ähnliches auf die Custom-Chips einrichtet, kann er den berühmten „Fireworks-DisplayMode" der Kickstart 1.0 betrachten, auf den die Chips dann automatisch umschalten. Maxon und der Autor übernehmen allerdings keinerlei Garantie auf Rechner, Monitor, Software, Gemütszustand des Users oder sonstiges!)
2 Das Objekt ist eine Konstante, die keinen Ort, sondern nur einen Wert hat: das vorzeichenbehaftete Offset. Leider lassen sich nicht alle LONG-Konstanten hiermit beschreiben!
3 Das Objekt liegt in einem Prozessorregister, Offset gibt an, welches Register: Offset 0-7 = D0-D7, 8-17 = A0-A7, 16-23 = FP0-FP7. Sollte Motorola auf die Idee kommen, im Usermode weitere Register einzuführen, so wird diese Liste (von Maxon Computer) verlängert werden. Von den Registern, die im Supervisor-Modus zugänglich sind, hat ein Compiler die Finger zu lassen, deshalb sind sie hier nicht aufgeführt.

4 Das Offset bezieht sich auf A6, zum Zugriff auf die statischen Daten von innerhalb einer Library.

5 Das Offset bezieht sich auf A7, falls eine Funktion keinen Stackframe aufbaut, aber trotzdem lokale Variablen oder Argumente auf dem Stack hat. Normalerweise ändert sich A7 aber ständig, so daß diese Methode eine gute Art darstellt, einen Debugger durcheinander zu bringen. (Ein sehr gut optimierender Compiler könnte mit sowas aber zurecht kommen.)

6 Das Offset bezieht sich auf A5, für normale lokale Variablen und Funktionsargumente auf dem Stack.

7 Das Offset bezieht sich auf A4, für das „small Data"-Speichermodell. Dieser Wert für $TT ist eigentlich nur bei Programmen sinnvoll, die voll reentrant sind und trotzdem sowas wie globale Variablen benötigen, die dann von jedem Prozeß neu „geAllocMem()t" und in der Folgezeit über A4 adressiert werden. Alle anderen Programme sollten -1 bis -128 (die Hunk-Nummern) benutzen, da der Debugger dann unabhängig vom derzeitigen Prozessorzustand die globalen Variablen anzeigen kann.

8 Das Objekt ist Teil einer Klasse, und sein Offset bezieht sich auf die Startadresse eines anderweitig bekannten Klassenobjekts.
Achtung: Funktionen, die Klassenmembers sind, haben einen Extra-Eintrag in ihrem Typ, der sie als zu eben dieser Klasse gehörig ausweist. Deshalb ist der Codeort einer solchen Member-Funktion der CODE-Hunk, außer sie ist inline, dann hat sie wohl keinen eindeutigen CodeOrt, und $TT muß 0 sein.

9 Dieses Objekt hat keinen Ort, es handelt sich nämlich um einen typedef-Bezeichner des anderswo angegebenen Typs. Offset muß 0 sein.

10 Der Ort dieses Objekts befindet sich nicht in diesem Modul (oder vielleicht doch?), da es (nur) als `extern` deklariert wurde, ein Debugger muß also alle seine geladenen Module durchsuchen, um die echte Adresse zu finden.

11 Dies ist ein Bitfield in einer Struktur. Offset wird in zwei Teile gespalten: $WWOOOO, wobei dann $WW die Breite des Feldes in Bits ist, und $OOOO der Offset (in Bits) des Feldes bezüglich der Startadresse der Struktur, und zwar so gemessen, wie ein 68020 auf ein Bitfield zugreift (Adresse ist die des MSB). $WW darf nur Werte von 0 bis 32 annehmen, wobei 0 sinnlos ist. Vor der leichtfertigen Verwendung von Bitfields kann auch an dieser Stelle nur gewarnt werden, sie blähen ein Programm schnell auf, anstatt es kürzer zu machen! Die Amiga-Includes z.B. verwenden nicht ohne Grund kein einziges Bitfield!

Index

Symbole

#?.mdbg 63
#?.mdbi 51, 63, 64
????? 37

A

About 39
Absturz 68
Achteck 53
Adresse 27
Adressraum 33
Adressregister 31
aktive Kachel 17
Aktuell 46
aktuelle Aufrufkette 40
aktuelle Fenster 47
aktuelle Funktion 25, 26, 40, 56
Aktuelle-Variablen-Fenster 23
aktuelles Fenster 17
aktuelles Verzeichnis 58, 70
Allgemein 45
AllocEntry 33
Amiga-C 60
AmigaDOS 70
angewählte Zeile 17
ANSI-C 60
Argumente 11, 28, 44
ASCII 32, 51
Assembler 18
Assemblerdebugger 60
Assemblermodus 14, 30, 32
Aufrufkette 25
Aufzählungstypen 37
Ausnahmebedingungen 19
automatisch geöffnet 58

B

Basisadresse 29
Basisklassen 28
BCPL-Programme 62
Bedingung 54, 55
beenden 39
Befehle 38

Benutzeroberfläche 68, 70
Betriebssystem 30, 72
Bildschirm 46
Binärformat 58
Breakpoints 44
BSS 22
Byte 33

C

Cache 71
Change 50
ChipMem 63
CHK 74
CLI 13
Coprozessoren 72
CPU-Register-Fenster 30
Custom-Chip-Register 67

D

DATA 22
Daten ändern 50
Datenbyte 33
Datenregister 31
Datenträgers 65
Datentyp 27, 37, 60
Debugger 11
Debuggerdatei 63
Deklarationen 21
Dezimal 32
Dialogfenster 34
Disable() 44, 64, 71
Disassemblerlisting 20, 30, 52
DIV 74
Division durch Null 73
Doppelklick 54, 55
Durchlaufzähler 57

E

einfachen Haltepunkt 54
Einstellungen 45
Einzelschritt 52
Einzelschrittmodus 53, 73, 74
Endlosschleife 43, 70
Enforcer 14, 59, 61, 68
enum 28, 77
Escape-Sequenz 37
Exception-Vektoren 75
exec/Tasks.h 31
exit() 66

F

Fachausdrücke 59
Feld 37
Fenster 39, 46
Fensternummer 17, 49
FFS 69
FILLPEN 54
FindTask 69
Fließkomma 36
Fließkommazahl 37, 72
Folge hinein 42
Forbid() 44, 71
Funktion 11
Funktionen-Fenster 21
Funktionsargumente 40
Funktionsaufruf 42
Funktionsrumpfs 55

G

Geschichte 34
Geschwindigkeit 42
global optimierenden Compiler 70
globale Menübaum 38
globale Variablen 21, 40
Globale-Variablen-Fenster 21
globalen Funktion 40
Gültigkeitsbereich 24
Guru 74

H

Haltepunkt 19, 43, 44, 53, 70
  Achteck 53
  hohl 54
  inaktiver 54
  löschen 57
  setzen 53
Haltepunkt löschen 57
Haltepunkt-Dialogfenster 54, 55
Hauptfenster 18, 48
Hauptspeicher 32, 41, 67
Hauptspeicherzugriffe 68
Hexdump 21
History 34
höchste Adressbit 33
hohl 54

I

IFF 77

Info 39
inline 69
Inspect-Fenster 27, 49
Inspiziere-Fenster 27
interaktive Hilfesystem HotHelp 39
Interrupt 10
into 42
Intuition 71
invertiert 32

K

Kachelgrößensymbol 16
Kacheln 16
Klartext 58
Klasse 37
Klassenobjekt 28
Kommandozeile 60
Kommandozeilenargumente 44
Kommandozeilenoption 58
Kommandozeilenoptionen 14
komplexen Haltepunkt 54
Konfiguration 45, 58
kontextsensitive Hilfe 38, 39

L

Langwort 33
Latin 1 33
Läuft: ready 19
Läuft: running 19
Läuft: waiting 19
Layers.library 48
LINK A5,#$xxxx-Maschinenbefehl 25
lokale Menüs 38, 48
Lokale-Variablen-Fenster 23
lokalen Variablen 70

M

main() 25, 39, 60, 66, 77
Maschinenbefehl 52
Maschinensprache 43
mathXXXX.librarys 72
Maus 36, 54
MaxonSDB.prefs 45, 58
Message 64
MMU 68
Modul 11, 40
Modul-Fenster 30, 62
Modulfenster 21
Modulname 30

Multitasking 10

N

Netzwerken 58
Neuzeichnen 46
Nummer 47

O

Operator 56
Originalgröße 46
over 42
Overlays 66

P

Pfad 30
Pfadverweis 63
privilegierter Befehl 74
Process 31
Programm laden 38
Programmiersprache 59
Programmstart 58
Programmzähler 19, 42, 52, 53
Programmzeile 53
Prozeß 10, 19, 70
Prozeßadresse 31
Prozessor-Exceptions 73
Prozessorregister 41
Prozessorstatusregisters CCR 31
Prozeßstruktur 31
PTEST 68

Q

Quellmodus 14
Quelltext 18, 52
Quelltextdebugger 60

R

R_FormAlert 62
Rahmen 25
rct.library 62, 65
Referenz 28
Referenzen 37
Register 30
Rekursion 56
Remove() 64
residente Libraries 62
Ressourcen 39

Rollbalken 16
ROM 43, 64, 67
Rückgabewert 28
Rückgabewerts 25
Rückkehr 43
ruhender Haltepunkt 56

S

SafeTrace 60, 74
Schachtelung 77
Schranke 55, 57
Schreibschutz 65
Schritt 42
Settings 45
Shell 13
SHINEPEN 54
sichtbaren Speicherausschnitt 33
Signale 64
Skalar 22, 27
Speicher 41, 46
Speicher-Fenster 32, 49
Sprache 59
Stack 40
Stack-Frame 25
Stack-Rahmen 70
Stapel 40
Stapelzeiger 31
Starten 44
Startup-Code 11, 60
Statusregister 31
Statuszeile 19, 62
STOP-Schild 54
Stopp 43
struct Process * 31
Supervisor 10
Supervisor-Modus 60

T

Tabulator 14, 59, 61
Taschenrechner 36
Tastatur 34
Tastaturkürzel 38
tc_TrapCode 73
Texteingabe-Fenster 34, 50
Texteingabefeld 35
TEXTPEN 54
Trace 42
Trace-Mode 74
Trace-Modus 60
TRAPV 74

TS_READY 64
TS_WAITING 64

U

Über 39
über 42
Überwachen 40
Überwachen-Fenster 24, 48
Umlaute 33
ungerade Adresse 34
UniCode 32, 33, 51
UNLK A5 25
Unterbereichstypen 37
Unterprogramm 43
ursprünglichen Adresse 33
User-Modus 30

V

VBR-Register 75
Verbund 37
verkettete Liste 49
Verzeichnis 45, 58
virtueller Speicher 14, 59, 61, 68
void* 29
Vorgänger-Funktion 26
Vorheriges 47

W

wack 68
Wait() 43, 64
Wait(0L) 70
WaitPort() 64
Wartezustand 64
wbmain() 78
wichtigsten Variablen 24
Wort 33

Z

Zähler 55, 57
Zeichensatz 33
Zeiger 37
Zeilennummer 19, 41
Zoom 46
zusammengesetztes Objekt 29
zweisprachig 38